A Novel Reseeding Mechanism for Improving Pseudo-Random Testing of VLSI Circuits

Jiann-Chyi Rau*, Po-Han Wu and Ying-Fu Ho

Department of Electrical Engineering, Tamkang University, Tamsui, Taiwan 251, R.O.C

Abstract

During built-in self-test (BIST), the set of patterns generated by a pseudo-random pattern generator may not provide sufficiently high fault coverage and many patterns can't detect fault (called useless patterns). In order to reduce the test time, we can remove useless patterns or change them to useful patterns (fault dropping). In fact, a random test set includes many useless patterns. Therefore we present a technology, including both reseeding and bit modifying (a.k.a. pattern mapping) to remove useless patterns or change them to useful patterns. When patterns changed, we pick out number of different fewer bits, leading to very short test length. Then we use an additional bit counter to improve test length and achieve high fault coverage. The technique we present is applicable for single-stuck-at faults. Experimental results indicate that complete fault coverage-100% can be obtained with less test time.

Key Words: BIST, LFSR, Pseudo-Random Testing, Reseeding

1. Introduction

As the IC design trends are migrating rapidly into the so called Systems-on-a-Chip (SoC) approach and various pre-designed and pre-validated cores are integrated in a chip, the complex designs are creating serious challenges for external Automated Test Equipment (ATE) and the built-in self-test (BIST) has emerged as a promising solution to the VLSI testing problem. BIST is a design for testability methodology aimed at detecting faulty components in a system by incorporating test logic on-chip. The main components of a BIST scheme are the test pattern generator (TPG), the response compactor, and the signature analyzer. The test generator applies a sequence of patterns to the circuit under test (CUT), the responses are compacted into a signature by the response compactor, and the signature is compared to a fault-free reference value. The Figure 1 shows architecture of the BIST.

Many digital circuits contain random-pattern-resistant (r.p.r.) faults that limit the coverage of pseudo-random testing [1], that is, circuits with such r.p.r. faults will have low detectability (few random patterns detect them). Several techniques have been suggested for enhancing the fault coverage achieved with BIST. These techniques can be classified as: (1) Modifying the circuit under test (CUT) by test point insertion [1,2], or by redesigning the CUT [3,4], to improve the fault detection probabilities. (2)*Weighted pseudo-random patterns*, where the ran-

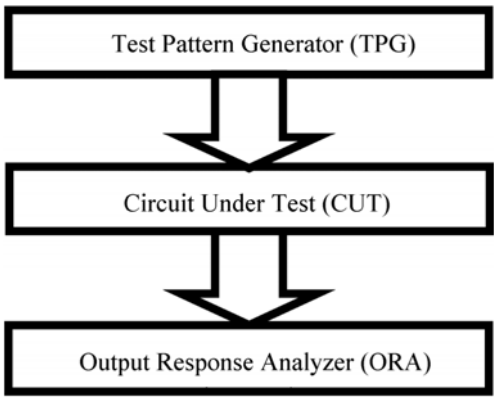


Figure 1. Architecture of the BIST.

*Corresponding author. E-mail: jcrau@ee.tku.edu.tw

dom patterns are biased using extra logic to increase the probability of detecting r.p.r. fault [5]. (3) Mixed-mode testing where the circuit is tested in two phases. In the first phase, pseudo-random patterns are applied. In the second phase, deterministic patterns are applied to target the undetected faults [6,7].

This paper uses an additional bit counter and modifying circuit in which deterministic test cubes are embedded in the pseudo-random sequence of bits. A procedure is described for designing the modifying-bit sequence generator and using an additional bit counter in a way that to decrease both test length and area overhead with obtaining high fault coverage. Our approach was due to addition modifying circuit and additional bit counter. It guarantees that certain test cube will be applied to the circuit-under-test during a specified test length. The Figure 2 shows the global operations.

In the proposed scheme of this paper, we just use a little storage for additional bit counter and that use less number of test length and seed are used to achieve the desire high fault coverage. The paper is organized as follows. Section II introduces the related literature. Section III describes the modifying-bit architecture, additional bit counter and the procedure for obtaining useful patterns which use fewer number of test length. Section IV shows the simulation results and Section V concludes the paper.

2. Related Work

In serial BIST, deterministic patterns are applied after a random testing to reduce number of the pattern. The deterministic pattern are loaded into the LFSR and then expended into the desired patterns in the scan chain.

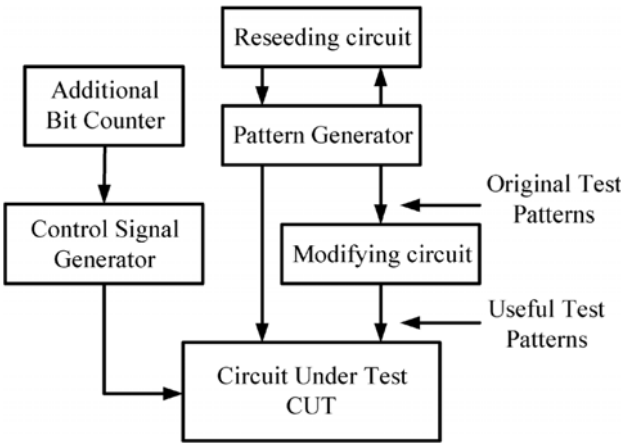


Figure 2. Block diagram for generation of useful patterns.

The work [8] presented a reseeding-based technique that improves the encoding efficiency by using variable-length seeds together with a multiple polynomial LFSR. The technique reuses part of the scan chain flip-flop in expanding the seeds.

In [9], random patterns that don't detect r.p.r. faults are mapped to ATPG generated cubes through combinational logic. The mapping is performed in two phases, the pseudo-random patterns are identified in the first step, and the ATPG cubes are loaded in the second step. Several iterative minimization heuristics are applied to reduce the area overhead of the mapping logic.

In [10], they loaded new seed by putting the LFSR in the state that precedes the seed value, so that at the next clock pulse, the new seed is in the LFSR, and their technique is based on deterministic seeds which expand into ATPG patterns so 100% fault coverage can be achieved. The algorithm they present is based on the following strategies: (1) generate ATPG patterns for faults that were not detected with pseudo-random patterns and calculate seeds for these patterns, (2) when a seed is loaded into the LFSR, let the LFSR run in autonomous mode for sometime because there is a chance that some of the ATPG patterns will drop more faults so that some of the ATPG patterns are not needed, (3) as long as pseudo-random patterns don't detect faults, the LFSR should be loaded with a new pattern.

The above schemes use seeds that don't particularly target undetected faults, so the test length would be increased. Our technique is based on deterministic seeds which expand into ATPG patterns so high fault coverage can be achieved and with smaller test length.

In [10], Reseeding refers to loading the LFSR with a seed that expands into a pre computed test pattern. The operation of the reseeding circuit is as follows: the LFSR starts running in autonomous mode for sometime according to the reseeding algorithm. Once it is time for reseeding, a seed is loaded into the LFSR, which then goes back to the autonomous mode and so on and so forth until the desired coverage is achieved. The new seed is in the LFSR. Their technique uses MUX between flip-flops as shown in Figure 6. By activating the select line of a given MUX, the logic value in the corresponding LFSR stage is inverted.

Figure 3 shows an example for a 4-stage LFSR connected to one scan chain with 10 flip-flops. This Figure

Figure 3. A 4-stage LFSR connected to a chain.

will be used as an example for illustrating the equation generation technique.

For every flip-flop in the scan chain, there is a corresponding equation in terms of the bits of the LFSR. Let's label the scan flip-flops by S_0 to S_{m-1} where *m* is the size of the scan chain. Also, let's label the stages of the LFSR by L_0 to L_{n-1} where *n* is the size of the LFSR. In the example above, the equations for the n most significant flip-flops of the scan chain are: $S_9 = L_3$, $S_8 = L_2$, $S_7 = L_1$, and $S_6 = L_0$ because after *n* clock cycles the bits of the seed end up in the most significant bits of the scan chain. The reader is invited to verify the remaining equations as Figure 4.

We can represent the above equations by an $m \times n$ matrix as Figure 5. In which the rows correspond to the LFSR stages and the columns correspond to the scan chain flip-flops. An entry (i,j) is 1 if and only if L_j appears in the equation of S_i. According to this system, the following matrix shows the equations for all the flip-flops in the scan chain of the example above:

$$S_5 = L_0 \oplus L_3 \qquad S_4 = L_0 \oplus L_2 \oplus L_3$$
$$S_3 = L_0 \oplus L_1 \oplus L_2 \oplus L_3 \qquad S_2 = L_1 \oplus L_2 \oplus L_3$$
$$S_1 = L_0 \oplus L_1 \oplus L_2 \qquad S_0 = L_1 \oplus L_3$$

Figure 4. The remaining equations of the scan flip-flops.

$$E = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{matrix} S_0 \\ S_1 \\ S_2 \\ S_3 \\ S_4 \\ S_5 \\ S_6 \\ S_7 \\ S_8 \\ S_9 \end{matrix}$$
$$L_0 \quad L_1 \quad L_2 \quad L_3$$

Figure 5. The matrix which represents the equations.

3. Our Embedding Algorithm

3.1 Modifying-Bit Architecture

In our technique, we are way of utilizing reseeding technique, and added to our method (i.e. modifying pseudo-random bit). The built-in reseeding [10] (encoding the seeds in hardware) refers to loading the pseudo-random pattern generator (PRPG) with a seed that expands into a pre-computed test pattern. The Figure 6 shows the architecture of the reseeding with modifying-bit circuit.

In order to reduce hardware of the modifying-bit logic (MBL), it is the most important to choose a pseudo-random sequence (i.e. some of the bits altered to specifying bits) from the pseudo-random pattern. The operation of the MBL was in the control of pattern counter and bit counter. The Figure 7 shows the circuit of modifying-bit logic.

When constructing the bit counter, the states of the bit counter can be decoded by simply using on n-input condition logic. The n is equal to the number of bits in the bit counter as shown in Figure 8, where the number of bit counter depends on length of the test cube. As for number of the bit counter, if the number of one test cube is equal to 100 bits so the number of bit counter equal to 7 bits.

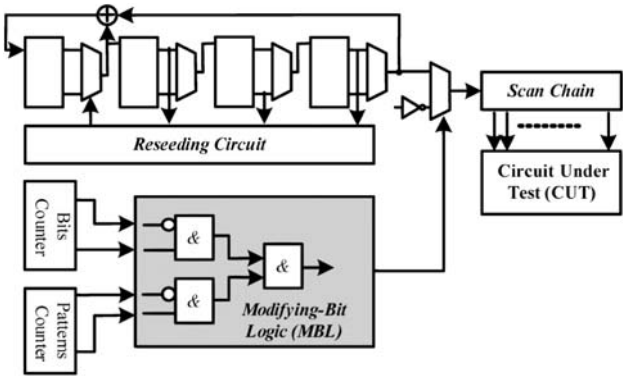


Figure 7. Circuit of modifying-bit logic.

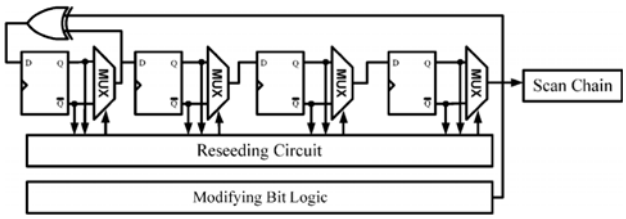


Figure 6. Stage reseeding with modifying circuit.

In new BIST architecture, a bit-counter function is used to choose when to bit value of LFSR is to be shifted into the scan chain; meanwhile, check on the position of the different bits values the correlation between useless pattern and test cube. Figure 8 shows the example will be used to illustrate the procedure described in our method. For example, the value of the 8th and the 9th bits of c2 want inversion to "1" while the pattern-counter equals c2 (pattern-counter = 0010) and the Bit-counter equals eight and nine. That is because other of test cube bits positions are the same as pseudo-random pattern c2 (undetected fault pattern) and so inversion 8th and 9th bit position of c2 at pseudo-random sequence, and that shifted into the scan chain in order to embed deterministic test cube in the sequence.

In the architecture, a bit counter used to choose when to change the bit value of the useless patterns (the different bits values the correlation between useless pattern and test cub). Our technique is based on modifying some bits on useless patterns of pseudorandom mode to shorten the test length and further the number of seeds. The Figure 9 shows Modifying-Bit Logic. We use n-input AND gates where n is equal to the number of bits in the bit counter, and the max number of bit counter depend on length of the test cube. We pay the price in hardware overhead. If the total amount of Modifying-bits is k, we just need k Modifying-Bit Logic.

The example of the Figure 10 obtaining test cubes shows as follows, the 3rd, 5th and 6th bit of useless pattern

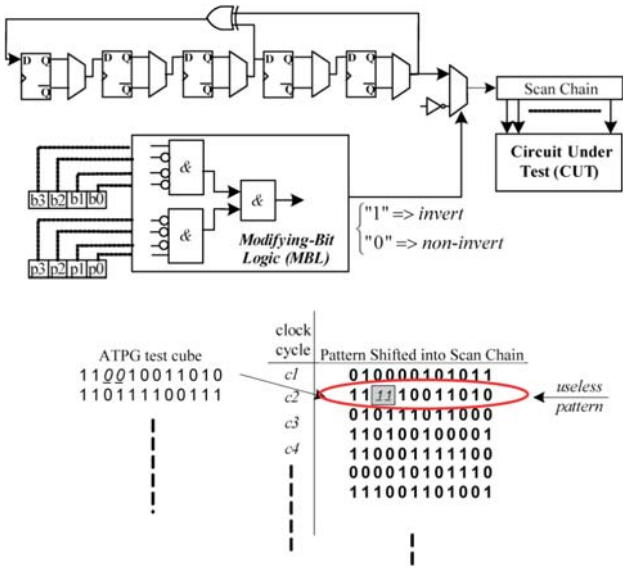


Figure 8. Modifying-bit example.

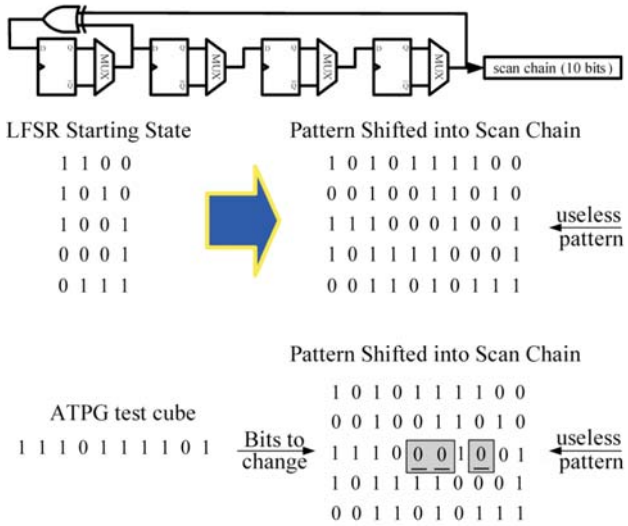


Figure 10. Obtaining the useful pattern.

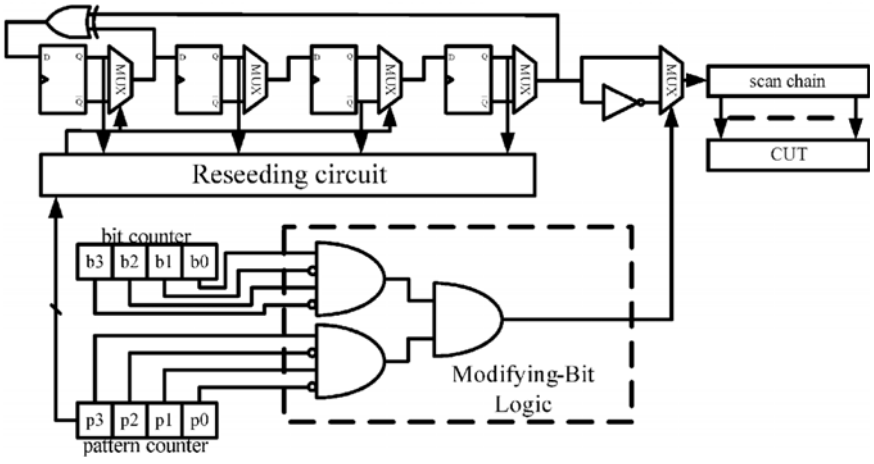


Figure 9. Architecture of modifying-bit.

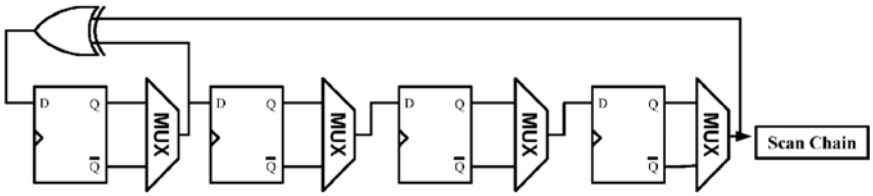
must be changed into "1" and so can generation ATPG test cubes when the condition established the pattern counter and the bit counter both. The other bits positions of the useless pattern are the same as ATPG test cubes and so only inversion position of the 3rd, 5th and 6th bit of the pattern at pseudorandom sequence of bits that is shifted into the scan chain in order to embed deterministic test cube in the sequence.

3.2 Additional Bit Counter

In the common logic BIST architecture, if we want to disable the "Scan Enable" signal for capturing, we can use a bit counter. Generally speaking, the bit counter loaded with the value that corresponds to the length of the scan chain for every pattern. The bit counter is decreased by 1 at each clock cycle. When the bit counter counts to zero, it means that the test pattern is loaded into the scan chains, and "Scan Enable" signal is disabled for one clock cycle, than we can capture in this time.

We need to load the bit counter register with different values corresponding to the number of cycles before the next capture, if we want to reach the desired seed in the pseudo-random pattern generator (PRPG). The value corresponds to the length of the scan chains plus the distance of the two useful patterns in the LFSR sequence.

As an example in Figure 11, assume that the patterns 1000 and 1110 are useful and the first useful pattern is in the LFSR, so we load the bit counter register with 6. After 4 clock cycles, the first pattern are loaded into the scan chain, and at clock cycle 6 (the length of the scan chains plus the distance of the two useful patterns), the second pattern are loaded into scan chain as shown in Figure 11. For another example as Figure 11, assume that the ATPG tool generate the following two patterns:



Clock cycle	LFSR sequence 4 - bit	Content of scan chain 4 - bit	Content of scan chain 10 - bit
0	1000	xxxx	xxxxxxxxxx
1	1100	0xxx	0xxxxxxxxx
2	1110	00xx	00xxxxxxxx
3	1111	000x	000xxxxxxx
4	0111	1000	1000xxxxxx
5	1011	1100	11000xxxxx
6	0101	1110	111000xxxx
7	1010	1111	111000xxx
8	1101	0111	01111000xx
9	0110	1011	101111000x
10	0011	0101	0101111000
11	1001	1010	1010111100
12	0100	1101	1101011110
13	0010	0110	0110101111
		(a)	(b)

Figure 11. Example of additional counter.

(X1XX1X1XX0, X10XXXX1XX). We can generate two seeds (1000, 1110) for the two patterns [11], so we load the bit counter register with 12. Assume the starting state of LFSR is 1000. After 10 clock cycles, the first pattern are loaded into the scan chain, and at clock cycle 12, the second pattern are loaded into scan, so the test length can be reduced (as long as the distance of the two useful patterns less than the length of the scan chain).

3.3 The Proposed Embedding Algorithm

In [10], Reseeding refers to loading the LFSR with a seed that expands into a pre computed test pattern. The operation of the reseeding circuit is as follows: the LFSR starts running in autonomous mode for sometime according to the reseeding algorithm. Once it is time for reseeding, a seed is loaded into the LFSR, which then goes back to the autonomous mode and so on and so forth until the desired coverage is achieved. The new seed is in the LFSR. Their technique use multiplexer (MUX) between flip-flops as shown in Figure 6. By activating the select line of a given MUX, the logic value in the corresponding LFSR stage is inverted.

In this paper, the design process is based on the following steps:

Step 1: ATPG tool is used to generate the test cubes and find the position (clock cycle) of the test cube run in pseudorandom mode.

Step 2: We must consider the waste of cycle (position of useless patterns) between the useful patterns at run pseudorandom mode. That is because position of useless patterns will be overwritten by test cube (change some bits). As long as pseudo-random patterns don't drop faults, the seed loaded into the LFSR that could be skipping some useful patterns, and therefore must be regenerate those patterns.

Step 3: If the distance of the two useful patterns is less than the length of the scan chain, we can use an additional bit counter to reduce the test length.

Step 4: The useless patterns of LFSR run in autonomous mode as compare with ATPG test cubes, and count number of different bits position and pick out number of different less bit from all test cubes by using C language in order to minimize hardware overhead.

Step 5: In Step 4, we can find the pattern that need to be changed, so we start to modify the position of the different bits at pseudorandom sequence of bits that is shifted into the scan chain.

Step 6: Lastly, if all ATPG test cubes whole appeared on pseudorandom mode, it means that all test cubes are embedded. Otherwise, loops back to step 4.

For a start, the step is to simulate the n-state LFSR for the given test length L to determine the set of pseudorandom patterns that are applied to the circuit-under-test. For each of the L patterns that are generated. Fault simulation is then performed on the CUT for the pseudorandom patterns to see which faults are detected and which are not. The faults that are not detected are the faults that require modifying of pseudorandom bit sequence.

Here, we must find out ATPG patterns was centered on the Nth cycle of pseudorandom mode, so that can choose suitable for the seed and then embedded into the LFSR (overwrite the original pseudorandom patterns). Generate ATPG patterns for faults that were not detected with pseudo-random patterns and calculate the seeds for these patterns.

When a seed is loaded into the LFSR, let the LFSR run in autonomous mode for sometime because there is a chance that some pseudo-random patterns will drop more faults so that some of the ATPG patterns are not needed (i.e. the pseudo-random pattern is able to detect faults in the pseudo-random mode, and therefore some of the ATPG patterns are not needed).

As long as pseudo-random patterns don't drop faults, the seed should loaded into the LFSR, yet the seed loaded into the LFSR and run in pseudo-random mode that maybe skipping some useful patterns, and therefore must be regenerate those patterns (i.e. the useful patterns was skipped from running the pseudo-random mode). If the useful patterns were skipped, we will be use modifying-bit techniques to regenerating for those patterns of skipping. The Table 1 shows the example will be used to illustrate the means.

The output of the reseeding circuit activates the select lines of the MUX to invert certain stages of the LFSR such that the desired seed is loaded in the next clock. As seen as Figure 12, the only modification the LFSR compared to a regular LFSR is that the LFSR flip-flops are replaced by multiplexed flip-flops just like the scan chain.

Table 1. Modifying-bit example

cycle	Q1 Q2 Q3Q4	cycle	Q1Q2Q3Q4
0	1 0 0 0	8	1 1 0 1
1	1 1 0 0	9	0 1 1 0
2	1 1 1 0	10	0 0 1 1
3	1 1 1 1	11	1 0 0 1
4	0 1 1 1	12	0 1 0 0
5	1 0 1 1	13	0 0 1 0
6	0 1 0 1	14	0 0 0 1
7	1 0 1 0	15	1 0 0 0

seed = 0100 = c12

select line active = c6 XOR c11

= 0101 XOR 1001

select lines of Q1 and Q2 activated

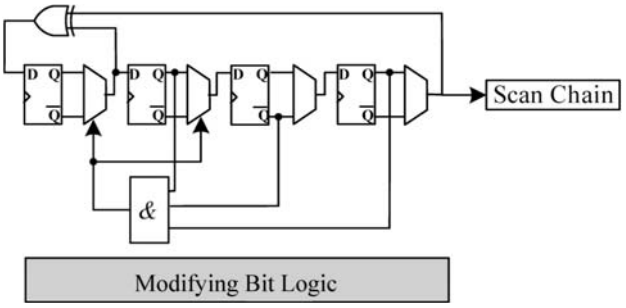


Figure 12. Hardware implementation.

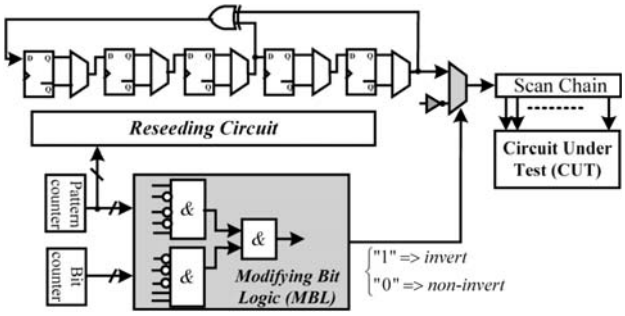


Figure 13. The MBL was control by counter.

Let's turn our attention to the reseeding circuit by looking at the following example. The Figure 12 is an example using a 4-stage self-reseeding LFSR (LFSR with reseeding logic) with a primitive polynomial. The Table 1 shows the full sequence of the regular LFSR. Assume that we want to reseed after the 6th cycle (c6). The reseeding circuit needs to be a condition logic that takes as inputs the contents of the LFSR at c6. So in the example the input to the reseeding & (condition logic, the & as Figure 13) is $\overline{Q1}Q2\overline{Q3}Q4$. All the cycles that are not part of the desired sequence can be used to minimize the reseeding circuit.

As an example, let the seed be 0100 (c12); we can easily calculate c11 given the polynomial of the LFSR (c11 = 1001). The reason we calculate c11 and not c12 is because we want the seed to be loaded into the LFSR in the next clock cycle. XORing c6 with c11 yields 1100 which means that the output of the reseeding & should activated the select lines of the MUX of Q1 and Q2.

If the seeds are required, every select line will activate it to complement the contents of its corresponding flip-flops, and then run pseudo-random mode.

The next step is to simulate the n-stage LFSR for the given test length L to determine the set of pseudo-random patterns that are applied to the CUT. For each of the L patterns that are generated, the starting n-bit state of the LFSR is recorded (i.e., the contents of the LFSR right before shifting the patterns into the scan chain). Fault simulation is then performed on the CUT for the pseudo-random patterns to seed which faults are detected and which are not. The pattern that drops each fault from the fault list is recorded. The faults that are not detected are the faults that require altering of pseudo-random bit sequence. The pseudo-random bit sequence must be altered to generate test cubes that detect the undetected faults. An automatic test pattern generation (ATPG) tool is used to obtain test cubes for the detected faults.

As show in Figure 13, there is an added MUX in the output side of the LFSR (i.e. gray MUX). The operation of the MUX was chosen by output value of the MBL. The output of the MBL equals 1, the output value of the LFSR will invert. If output of the MBL equals 0, the output value of the LFSR don't change it, and that the sequence shift into the scan chain.

Lastly, if all ATPG test cubes appeared on scan chain, no other than, all test cubes that are embedded, otherwise, loop back to the last step.

The modifying-bit generator is designed so that when the pseudo-random patterns undetected faults that are altered to embed test cube are added to the set of patterns that drop faults.

The Figure 14 shows the circuit of Modifying-Bit design flow. In our technique deterministic test cubes were generated using the ATPG tool of the SIS and the synthesis tool used the Synopsys® Design Analyzer and simulator of the Verilog-XL. Also, we have used Verilog Hardware Description Language (Verilog HDL) for structure of the LFSR.

4. Simulation Result

In this section we present the results of some simulation experiments. We performed our experiments on some of the ISCAS 89 benchmarks. The characteristics of the benchmarks we used are shown in Table 2. The Table 3 shows the number of primary inputs, number of faults, total amount of modify-bits and flip-flop in the scan chain. The BC column lists the sizes of the bit counters and the ABC column lists the sizes of the additional bit counter. The number of changing bits determines the area of the modifying circuit. In our technique, deterministic test cubes were generated using the ATPG tool of the SIS.

We performed some simulation experiments to compare our technique with [12]. The reseeding with modifying-bit generators were designed to provide 100% fault coverage of all detectable single stuck-at faults for a test length of fewer patterns. The experiment was designed such that pseudorandom patterns are applied first. Then, test patterns are generated and the seeds are calculated, and that the technique includes modifying-bit from the test patterns. Table 3 shows the test length, fault coverage and number of seed when our technique is used. The number of seed decrease ranged from 26.7% to 57.1%. The test length column reports the chosen length of the generated test sequence (Length).

5. Conclusion

We presented a built-in modifying-bit scheme based on change some bits of pseudorandom patterns in an

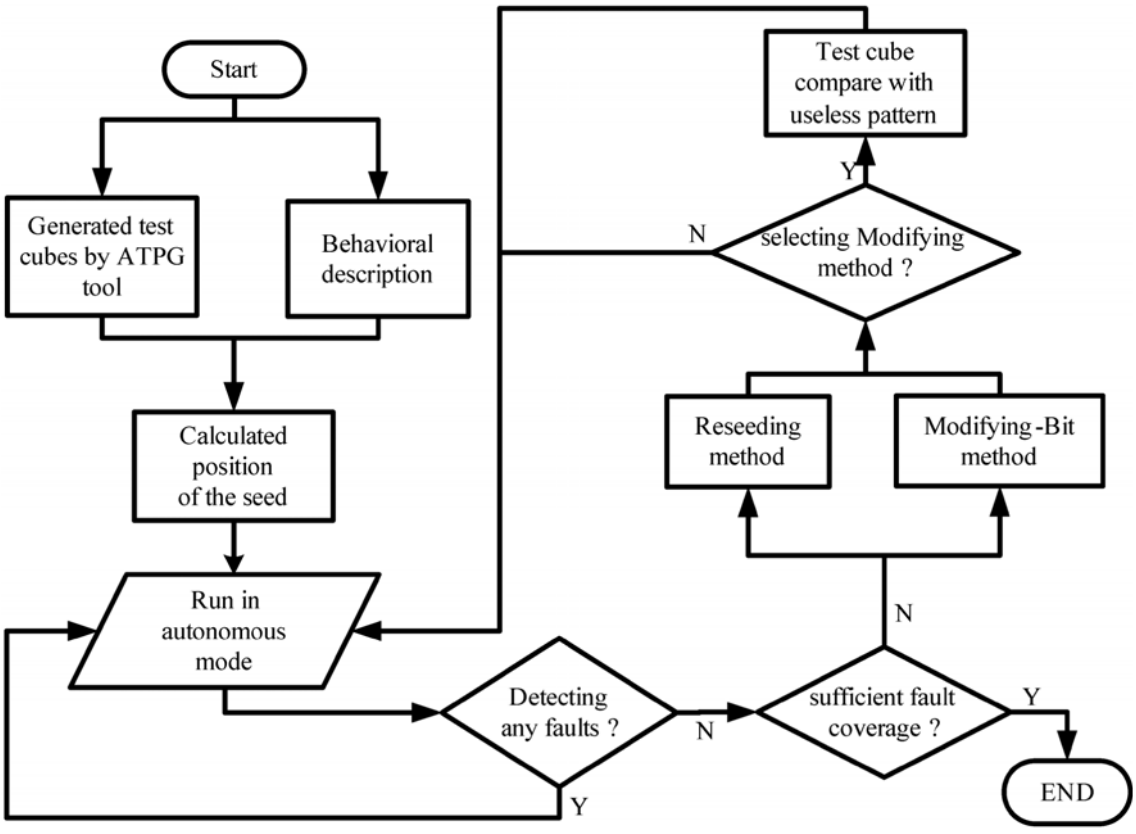


Figure 14. Modifying-bit design flow.

Table 2. ISCAS 89 circuits used in the experiments

Circuit	Inputs #	FF #	Faults #	BC	ABC	ABC USE	Amount of modifying bits
S420_89	19	16	840	6	7	3	130
S641	35	19	1274	6	7	4	652
S838_89	35	32	1676	7	8	3	405
S1494	8	6	2972	5	6	4	92
S9234	36	211	17350	9	10	2	5924

Table 3. Comparison of our technique and [6]

Circuit	Test Length		Use seed #			FC (%)	
	[6] calculating efficient seeds	Our	[6]	Our	Reduction (%)	[6]	Our
S420_89	1,000	852	52	32	38.5	94.0	100.0
	10,000					96.4	
S641	1,000	1,656	15	11	26.7	98.4	100.0
	10,000					99.5	
S838_89	1,000	3,412	95	65	31.6	85.4	100.0
	10,000					86.2	
S1494	1,000	712	14	6	57.1	99.1	100.0
	10,000					100.0	
S9234	1,000	16,256	598	415	30.6	84.9	100.0
	10,000					92.8	

on-chip. We are using not many hardware for the complete fault coverage and decrease test length without any external tester. Our structure of the MBL allows the designer to tradeoff between the number of seeds and the amount of modifying-bit logic.

Our technique uses reseeding the LFSR with just a few choose fit seed to generate some of the least correlated test cubes that require modifying-bit to embed. Our algorithm is very particular about what seed choose and compare test cube with patterns of modified bits selection. The simulation result shows that the numbers of seeds and test length have to be decreased when we pay the price in hardware overhead.

We presented a scheme include built-in modifying-bit and additional bit counter. High fault coverage (single-stuck-at fault) can be achieved with our technique without any external testing. We pay the price in hardware overhead in order to decrease test length. Our scheme allows the designer to trade off between the number of seeds and the amount of modifying-bit logic.

References

[1] Eichelberger, E. B. and Lindbloom, E., "Random-Pattern Coverage Enhancement and Diagnosis for LSSD Logic Self-Test," *IBM Journal of Research and Development,* Vol. 27, pp. 265–272 (1983).

[2] Touba, N. A. and McCluskey, E. J., "Test Point Insertion Based on Path Tracing," *Proc. of VLSI Test Symposium,* pp. 2–8 (1996).

[3] Chiang, C. H. and Gupta, S. K., "Random Pattern Testable Logic Synthesis," *Proc. of International Conference on Computer-Aided Design (ICCAD),* pp. 125–128 (1994).

[4] Touba, N. A. and McCluskey, E. J., "Automated Logic Synthesis of Random Pattern Testable Circuits," *Proc. of International Test Conference,* pp. 174–183 (1994).

[5] Eichelberger, E. B., Lindbloom, E., Motica, F. and Waicukauski, J., "Weighted Random Pattern Testing Apparatus and Method," US Patent 4,801,870, Jan. 89.

[6] Hellebrand, S., Reeb, B., Tarnick, S. and Wunderlich, H.-J., "Pattern Generation for a Deterministic BIST Scheme," *Proc. of International Conference on Computer-Aided Design (ICCAD),* pp. 88–94 (1995).

[7] Touba, N. A. and McCluskey, E. J., "Altering Bit Sequence to Contain Predetermined Patterns," US Patent 6,061,818, May (2000).

[8] Rajski, J., Tyszer, J. and Zacharia, N., "Test Data De-

compression for Multiple Scan Designs with Boundary Scan," *IEEE Transactions on Computers,* Vol. 47, pp. 1188−1200 (1998).

[9] Touba, N. A. and McCluskey, E. J., "Synthesis of Mapping Logic for Generating Transformed Pseudo-Random Patterns for BIST," *Proc. of International Test Conference,* pp. 674−682 (1995).

[10] Al-Yamani, A. A. and McCluskey, E. J., "Built-In Reseeding for Serial BIST," *Proc. VLSI Test Symposium,* pp. 63−683 (2003).

[11] Al-Yamani, A. A., Mitra, S. and McCluskey, E. J., "BIST Reseeding with Very Few Seeds," *Proc. VLSI Test Symposium,* pp. 69−74 (2003).

[12] Fagot, C., Gascuel, O., Girard, P. and Landrault, C., "On Calculating Efficient LFSR Seeds for Built-In Self Test," *Proc. of European Test Workshop,* pp. 7−14 (1999).

Manuscript Received: Apr. 11, 2006
Accepted: Aug. 13, 2007