Bit-Flipping BIST

Hans-Joachim Wunderlich Gundolf Kiefer

Computer Architecture Lab
University of Stuttgart, Breitwiesenstr. 20-22
D-70565 Stuttgart, Germany

Abstract

A scan-based BIST scheme is presented which guarantees complete fault coverage with very low hardware overhead. A probabilistic analysis shows that the output of an LFSR which feeds a scan path has to be modified only at a few bits in order to transform the random patterns into a complete test set. These modifications may be implemented by a bit-flipping function which has the LFSR-state as an input, and flips the value shifted into the scan path at certain times. A procedure is described for synthesizing the additional bit-flipping circuitry, and the experimental results indicate that this mixed-mode BIST scheme requires less hardware for complete fault coverage than all the other scan-based BIST approaches published so far.

Keywords: *Mixed-Mode BIST*

1. Introduction

Built-in self-test (BIST) is one of the most important techniques for testing large and complex systems. The efficiency of a BIST implementation is characterized by the test length and the hardware overhead required to achieve complete or sufficiently high fault coverage.

In a "test per scan" scheme, test registers feed and evaluate a (partial) scan path (see figure 1). In a "test per clock" scheme, some system registers are enhanced such that they generate patterns or compact test responses in a special test mode.

BIST schemes may be classified with respect to the kind of patterns they generate. Random patterns are most easily generated using linear feedback shift registers (LFSR) for scan-based BIST [EiLi83, BMS87], the multi-functional test registers for a "test per clock" scheme are somewhat more sophisticated [KMZ79, OWM87]. If the fault coverage of random patterns is not sufficient, weighted random patterns may be applied by a "test per scan" scheme [WLEF89, StWu91] or by a "test per clock" scheme [Wu87, Brgl89]. Even pseudo-exhaustive test sets can be generated by both methods [HWH90, BCR83].

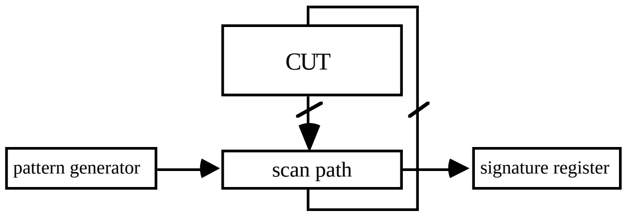

Figure 1: "Test per scan" scheme

In general, combinational circuits are not pseudo-exhaustively testable, and deterministic test sets have to be applied if the circuit is not allowed to be segmented by test points for timing or area reasons. A deterministic "test per clock" scheme may be implemented by designing an appropriate feedback function of a non-linear feedback shift register [DaMu81], or by including additional circuitry between an LFSR and the CUT which maps random patterns to deterministic test patterns [AkJa89, DUFA95, ToMc95b, ChPr95]. The first solution is only feasible for small circuits and test sets, and the second one slows down performance, as the additional test circuitry is part of the data path. Moreover, some effort is required to show that the test circuitry is fault free, too.

"Test per scan" schemes do not affect the system behavior so much, as only a scan path is included in the mission logic [KOEN91, HELL92, HELL95]. Usually, the deterministic patterns are applied after a random test to reduce the number of patterns and the hardware overhead. The most efficient way for

This work has been supported by the DFG under grant Wu 245/1-1, and was performed mainly at the University of Siegen, Germany.

implementing a scan-based mixed-mode test known so far is the approach based on special test sets and reseeding of multi-polynomial LFSRs as presented in [HELL95].

In this paper, we present a mixed-mode "test per scan" scheme which is based on the fact that a random test set contains mostly useless patterns [ToMc95a] which can be transformed to a useful pattern by flipping just a few bits. This results in a structure as shown in figure 2.

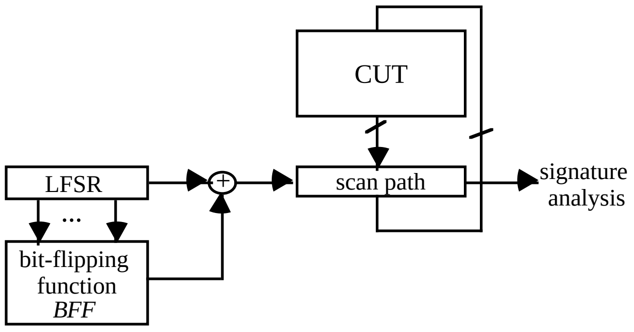

Figure 2: Bit-flipping BIST

The bit-flipping function *BFF* has a very small off-set which corresponds to the useful random patterns, a very small on-set corresponding to bits to be flipped, and a very large don't-care-set. This results in a large potential for optimization which will be exploited systematically in the rest of the paper.

In the next section we compute expectation values of the number of bits to be flipped. In section 3, an efficient way for determining the bit-flipping function *BFF* is presented, and a synthesis procedure is proposed. The experimental results of section 4 show that the presented approach leads to solutions which are more efficient than the schemes previously proposed.

2. Number of bits to be flipped

The efficiency of the basic structure of figure 2 is due to the fact that not all bits of deterministic test patterns are specified. Usually they contain a very large number of don't-care bits to be used for optimizations [HELL95]. In the sequel, we estimate the number of bits of a random pattern which must be flipped in order to be compatible with an incompletely specified deterministic pattern.

Assume a scan path with n flip-flops, and an LFSR generating the pseudo-random test set M of cardinality $m := |M|$. Let T be a deterministic pattern with s specified bits and $n-s$ unspecified bits. The probability that there is a pattern $T_d \in M$ which has a conflict with T in at most d bit positions, $d \le s$, is estimated by

$$P_d \approx \frac{m}{2^n} t_d, \text{ where } t_d = 2^{n-s} \sum_{i=0}^{d} \binom{s}{i}, \qquad (1)$$

while $m \cdot t_d < 2^n$. For $m \cdot t_d \ge 2^n$ the probability is nearly 1. The term t_d denotes the absolut number of patterns which have a conflict with T in at most d bit positions. Formula (1) can be transformed into

$$P_d \approx \frac{m}{2^s} \sum_{i=0}^{d} \binom{s}{i},$$

and the expectation value of the number d of bits to be flipped depends on m and s:

$$E(m,s) = \sum_{d=1}^{s} d \cdot (P_d - P_{d-1}). \qquad (2)$$

Table 1 shows the expectation values for different random test sizes m and numbers of specified bits s.

m	$s=10$	$s=20$	$s=30$	$s=40$	$s=50$	$s=60$	$s=70$
1,000	0.02	2.78	6.09	9.54	13.32	17.17	21.11
10,000	0.00	1.79	4.66	7.83	11.39	15.03	18.74
100,000	0.00	0.90	3.53	6.50	9.65	13.19	16.64
1,000,000	0.00	0.05	2.54	5.21	8.29	11.52	14.89

Table 1: Expected number *E(m,s)* of bits to be flipped

As an example, for a pattern with $s = 20$ specified bits we can expect to find one out of 10,000 random patterns which has to be flipped at only two (≈1.79) positions. In general, the expected number of bits to be flipped in order to generate a precomputed test pattern is significantly less than the number of bits specified in that pattern.

3. Determining the bit-flipping function *BFF*

The bit-flipping function is constructed iteratively. In each step, it is enhanced, such that new deterministic patterns are contained in the output of the resulting pattern generator while certain old patterns remain unchanged.

For generating a test pattern, the bit-flipping function has a large don't-care set which can be used for minimizing the logic of the function. This way many of the useless patterns are modified, too, increasing the chance of detecting some additional faults. Sometimes these modifications have to be reverted. The best way to do so is another XOR gate, and the general form of a bit-flipping BIST structure is shown in figure 3.

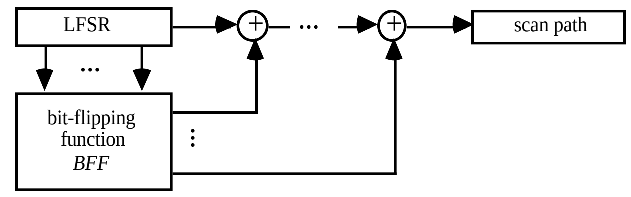

Figure 3: General form of bit-flipping BIST

In order to describe the synthesis procedure in a formal way we use the following variables (some of them have already been defined):

n	length of the scan path
m	number of patterns, test length
l	length of the LFSR
$S \subset \{0,1\}^l$	set of states of the LFSR during testing
$S_p^i \in S$	state of the LFSR while bit $p[i]$ of pattern $p \in \{0,1\}^n$ is generated
$S_p = \left\{ S_p^i \mid 1 \le i \le n \right\}$	set of states of the LFSR during generating the pattern $p \in \{0,1\}^n$
XOR	set of XOR-gates inserted between the LFSR and the scan path
F	set of all non-redundant faults

$BFF = (BFF_{x_1}, ..., BFF_{x_{|XOR|}})$, $x_1, ..., x_{|XOR|} \in XOR$, is being constructed incrementally, beginning with $BFF^0 \equiv 0$ and ending with BFF^R which provides complete fault coverage. For each iteration r, $0 \le r < R$, there is a set F_{hard}^r of faults which are not detected by BFF^r.

A boolean function can be uniquely defined by a set of product terms or its on-set (implying that the off-set is the complement of the on-set). In the following, we will use the symbol BFF_x^r for any of those representations, depending on the context.

3.1. The fix-set of BFF^r

In order to improve BFF^r, it is necessary to protect some patterns required for detecting a set of "critical" faults F_{crit}^r. Given F_{crit}^r, all patterns generated by the LFSR and BFF^r are simulated in several permutated orders, until a small subset $P = \{p_1, ..., p_k\}$ of patterns is found which still detects all faults in F_{crit}^r. In order to guarantee complete fault detection not all the bits of p_i, $1 \le i \le k$, need to be specified. Based on pessimistic 3-valued fault simulation, P is transformed into a set of patterns $P' = \{p'_1, ..., p'_k\}$ that contain as many don't-cares as possible and still detect all faults in F_{crit}^r. Let p'_i be one of these patterns and S_{p_i} be the corresponding set of states of the LFSR. The set of "fixed" states corresponding to p'_i is

$$FIX(p'_i) := \left\{ S_{p_i}^j \,\middle|\, \text{the } j\text{-th bit of } p'_i \text{ is specified} \right\}$$

and the entire fix-set is defined by

$$FIX^r := \bigcup_{i=1}^{k} FIX(p'_i).$$

Example. Assume we have a scan path of length $n = 5$ which is fed by the LFSR sketched in figure 4. The test length is $l = 5$ and $BFF^0 \equiv 0$. Table 2 shows the state sequence of the LFSR. The resulting pseudo-random patterns and the corresponding states are listed in table 3.

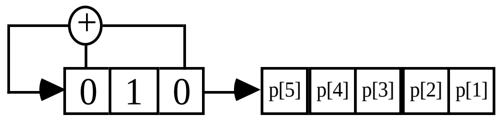

Figure 4: LFSR used in the example

s_0	010
s_1	001
s_2	100
s_3	110
s_4	111
s_5	011
s_6	101
$s_7 = s_0$	010
...	...

Table 2: States of the LFSR

#	pattern p[1]...p[5]	states
1	01001	s_0, s_1, s_2, s_3, s_4
2	$\underline{11}010$	$\underline{s_5}$, $\underline{s_6}$, s_0, s_1, s_2
3	$\underline{0}11\underline{1}0$	$\underline{s_3}$, s_4, s_5, $\underline{s_6}$, s_0
4	10011	s_1, s_2, s_3, s_4, s_5
5	10100	s_6, s_0, s_1, s_2, s_3

Table 3: Pseudo-random patterns and corresponding LFSR states

Let F_{crit}^0 be such that all faults in F_{crit}^0 can be detected by the patterns "11---" and "0--1-". The procedure for extracting essential patterns returns patterns 2 and 3, $P = \{p_1, p_2\} = \{ 11010, 01110 \}$, the analysis of essential bits transforms P to $P' = \{p'_1, p'_2\}$ = { 11---, 0--1- }. Remembering that p'_1 corresponds to the 2nd and p'_2 to the 3rd pattern, we can use table 3 to look up the fix-sets:

$$FIX(p'_1) = \{s_5, s_6\}$$
$$FIX(p'_2) = \{s_3, s_6\}$$

$$FIX^0 = FIX(p'_1) \cup FIX(p'_2) = \{s_3, s_5, s_6\} = \{110, 011, 101\}$$

3.2. Mapping test patterns to random patterns

Assume we have already defined the function BFF^r, and P is the set of patterns generated by the LFSR and BFF^r. Let T be the set of partially specified deterministic patterns which cover all faults in F^r_{hard}.

Now we have to select a test pattern $t_0 \in T$ and a random pattern $p_0 \in P$ such that t_0 can be mapped to p_0 efficiently.

Test patterns with only a few specified bits correspond to faults that are comparably "easy to detect" and might be detected by random patterns in some later iteration of the algorithm. So, $t_0 \in T$ is selected such that the number or specified bits is maximum.

Now a random pattern that can be modified has to be found. For any pseudo-random pattern p, let $off(t_0, p) \subset S_p$ be the set of LFSR states which correspond to bits equal to the bits specified in t_0, and let $on(t_0, p) \subset S_p$ be the set of states generating bits which are incompatible to the corresponding bits of t_0. The pattern t_0 can only be mapped to p if $on(t_0, p) \cap FIX^r = \emptyset$ holds.

The cost for assigning t_0 to p is estimated by the increase of the number of terms required for a 2-level implementation of BFF^r. We say, a minterm $c \in on(t_0, p)$ can be "efficiently expanded" and therefore does not cause any new product term if there is a $c_0 \in \bigcup_{x \in XOR} BFF^r_x$ such that

$$\left(FIX^r \cup off(t_0, p)\right) \cap \left(Expand(c, c_0) \setminus \{c, c_0\}\right) = \emptyset,$$

where the term $Expand(c, c_0)$ denotes the smallest boolean subspace covering both c and c_0 as used in ESPRESSO [BRAY84].

The cost of an assignment is estimated by

$$cost(t_0, p) = \left|\left\{c \in on(t_0, p) \mid c \text{ cannot be eff. expanded}\right\}\right|,$$

and a $p_0 \in P$ is selected for mapping such that $cost(t_0, p_0)$ is minimal.

Finally, the terms in $on(t_0, p_0)$ are assigned to XORs in a way that as many terms as possible can be efficiently expanded. During the XOR-assignment, no $c \in on(t_0, p_0)$ can be assigned to $x \in XOR$ if $BFF^r_x(c)$ is already 1.

3.3. The algorithm

After initializing $BFF^0 \equiv 0$ and determining the set F of all non-redundant faults, the following steps are repeated until complete fault coverage is achieved, r being the number of the current iteration:

1) Compute F^r_{hard} by fault simulation.

2) Compute FIX^r based on faults that have occured as hard faults in previous iterations, $F^r_{crit} := \bigcup_{i=0}^{r-1} F^i_{hard}$.

3) $BFF^{red} := Reduce_{FIX^r}(BFF^r)$.

   For each $x \in XOR$, an ESPRESSO-like REDUCE operation respecting the on-set $on := FIX^r \cap BFF^r_x$ is performed on BFF^r_x. Every term in BFF^r_x is reduced such that it contains as many specified bits as possible while the resulting function BFF^{red}_x still covers on.

4) Find a test pattern mapping consisting of a deterministic test pattern t_0, a pseudo-random pattern p_0, and an XOR-assignment $xor: on(t_0, p_0) \to XOR$.

5) For each $x \in XOR$:
   $BFF^{asgn}_x := BFF^{red}_x \cup \left\{c \in on(t_0, p_0) \mid xor(c) = x\right\}$,
   $FIX^{asgn} := FIX^r \cup on(t_0, p_0) \cup off(t_0, p_0)$.

6) $BFF^{r+1} := Expand_{FIX^{asgn}}(BFF^{asgn})$.

   For each $x \in XOR$, an ESPRESSO-like EXPAND operation respecting the off-set $off := FIX^{asgn} \setminus BFF^{asgn}_x$ is performed on BFF^{asgn}_x. Every term is expanded such that it contains as many don't cares as possible without producing a non-empty intersection with off.

Steps 4 and 5 can be repeated several times ("small loop"), thus avoiding expensive simulations and logic minimization procedures after every mapping. There is a trade-off between computation time (small for many small loop iterations) and the quality of the result (in general better if there are only a few assignments per simulation). In our experiments we increased the number of assignments with the number of iterations done so far.

Example. In order to continue the example of section 3.1 we assume a test pattern t_0 = 11-01 has been selected for mapping to any of the five pseudo-random patterns. Using the information of table 3, the sets $on(t_0, p)$ and $off(t_0, p)$ of states in which the bit-flipping function must be active or must not be active can be derived. For every pattern the condition

$on(t_0, p) \cap FIX = \emptyset$ is checked and $cost(t_0, p)$ is computed. Table 4 shows the results.

#	p	$on(t_0, p)$	$off(t_0, p)$	$cost(t_0, p)$
1	01001	s_0	$s_1,\ s_3,\ s_4$	1
2	11010	$s_1,\ s_2$	$s_5,\ s_6$	2
3	01110	$s_0,\ s_3,\ s_6$	s_4	∞
4	10011	$s_2,\ s_4$	$s_1,\ s_5$	2
5	10100	$s_0,\ s_3$	$s_3,\ s_6$	∞

Table 4: Finding a pattern for mapping t_0 = 11-01

Patterns 3 and 5 cannot be selected for mapping without violating the condition $on(t_0, p) \cap FIX = \emptyset$. The "cheapest" way of mapping t_0 is to modify pattern 1, so the on-set of this pattern p_0 is added to the bit-flipping function and the fix-set is updated:

$$BFF^{a\,\mathrm{sgn}} := BFF^0 \cup on(t_0, p_0) = \{s_0\} = \{010\}$$

$$\begin{aligned} FIX^{asgn} := FIX^r \cup on(t_0, p_0) \cup off(t_0, p_0) \\ = \{s_0, s_1, s_3, s_4, s_5, s_6\} \\ = \{\ 010,\ 001,\ 110,\ 111,\ 011,\ 101\ \} \end{aligned}$$

Finally, the bit-flipping function is expanded in a way that none of the terms in

$$FIX \setminus BFF^{asgn} = \{\ 001,\ 110,\ 111,\ 011,\ 101\ \}$$

is covered:

$$BFF^1 := Expand_{FIX^{asgn}}(BFF^{asgn}) = \{\ 0-0\ \}$$

Figure 5 shows the corresponding pattern generator including an implementation for the bit-flipping function BFF^1. The set of patterns produced by the new generator differs considerably from the original one (table 5). Nevertheless, patterns 2 and 3 are still compatible with the fixed patterns 11--- and 0--1-, and pattern 1 is now compatible with the deterministic test pattern t_0=11-01.

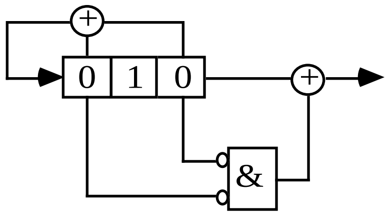

Figure 5: New pattern generator including bit-flipping logic

#	old	new
1	01001	$\underline{11}0\underline{01}$
2	$\underline{11}010$	$\underline{11}110$
3	$\underline{0}11\underline{1}0$	$\underline{0}11\underline{1}1$
4	10011	10011
5	10100	11100

Table 5: Old and new set of patterns

4. Experimental results

A series of experiments has been performed to determine the trade-offs between the length of the LFSR, the number of random patterns, and the area required for the mapping logic. The results are compared with the method of [HELL95] which provides the most area-efficient solution up to now.

Example circuits are those of the ISCAS-85 and combinational ISCAS-89 benchmarks [Brgl85, Brgl89] which still have undetected non-redundant faults after applying 10,000 random patterns.

The first two columns of table 6 show the circuit names and the number n of primary inputs. The next three columns are the reseeding results of [HELL95] where "LFSR" denotes the number of flipflops of the LFSR, "ROM" denotes the number of bits to be stored in a ROM and "Area" is the area required to implement this ROM and the flipflops of the LFSR using 1µm technology. The results of the presented bit-flipping approach are shown in the last four columns. Again, first the length of the LFSR is shown, the number of XOR-gates inserted between the LFSR and the scan path follows. Then the number of product terms required for a 2-level implementation is listed. The last column shows the area required for a PLA implementation including LFSR flipflops.

The results are based on 10,000 random patterns, and for both methods a complete coverage of all non-redundant faults is obtained. The length of the LFSR required for bit-flipping is much less than the length of the LFSR for reseeding. Moreover, the reseeding approach needs a multi-polynomial feedback function not counted in the table.

We compared the area of a ROM for reseeding with the area of PLA implementation for the bit-flipping approach. In many cases, the area for the bit-flipping approach is just 20% of the reseeding area, in one case it is 91%, but in all the cases there are distinct savings.

The bit-flipping approach has a considerable trade-off between hardware overhead and test length which seems to be in contradiction to the observation that random pattern testing leads to a saturation of fault coverage after a certain point [Wu85]. For selected examples, table 7 shows the number of undetected faults after a pseudo-random test and the number of

		Reseeding [HELL95]			Bit-flipping			
Circuit	n	LFSR	ROM	Area [μm^2]	LFSR	XORs	Terms	Area [μm^2]
s420	34	20	250	343,640	14	1	4	63,394
s641	54	22	183	344,013	14	1	3	62,544
s713	54	22	183	344,013	14	1	3	62,544
s838	66	36	1,623	533,077	14	2	37	99,566
s953	45	15	141	307,833	14	1	3	62,544
s1196	32	17	267	334,501	14	2	6	66,733
s1238	32	17	249	331,909	14	1	4	63,394
s5378	214	27	726	423,145	14	2	19	80,581
s9234	247	61	6,923	944,284	22	3	298	544,153
s13207	700	24	3,570	730,298	14	2	123	192,930
s15850	611	46	6,528	918,034	14	3	241	331,046
s38417	1,664	91	24,283	1,896,450	24	3	985	1,732,798
s38584	1,464	70	3,406	769,958	26	3	266	576,738
c2670	157	60	3,412	733,882	14	3	194	278,850
c7552	206	100	5,241	987,284	14	3	406	517,020

Table 6: Results of the bit-flipping method and reseeding after 10,000 random patterns

product terms required to achieve complete fault coverage for 1,000, 10,000 and 100,000 random patterns. In order to rule out the impact of the feedback polynomial, in all cases the same LFSR of length 32 was used, which leads to numbers different from those of table 6.

Circuit	$m = 1,000$		$m = 10,000$		$m = 100,000$	
	$\left\|F^0_{hard}\right\|$	Terms	$\left\|F^0_{hard}\right\|$	Terms	$\left\|F^0_{hard}\right\|$	Terms
s420	92	23	58	5	9	3
s641	18	12	8	6	5	4
s713	18	12	8	6	5	4
s838	425	171	340	109	259	90
s953	129	11	8	3	-	-
s1196	127	27	18	7	-	-
s1238	143	34	17	8	-	-
s5378	193	76	46	16	40	2
c2670	220	222	209	163	183	112
c7552	489	387	309	264	158	140

Table 7: Literals required for bit-flipping with different test lengths

There is still a decrease of the number of literals for the bit-flipping logic even for large test sets. This is due to the fact that not only the set of hard faults left for deterministic testing is reduced but also the expected number of bits to be flipped, and the conflicts during logic synthesis decrease.

The maximum CPU time required to obtain the results stated in table 7 using a Sparc10 workstation is in the order of some hours. The complexity of the most time consuming procedures (computation of the fix-set and pattern assignmnent) is linearly dependent on the test length m. Nevertheless, increasing m by a factor of 10 does not result in an increase of CPU time by the same factor. This is mainly due to the fact that for a larger test length fewer assignments and fewer iterations are necessary.

5. Conclusions

A new method for implementing a mixed-mode built-in self-test (BIST) has been presented. The new scheme requires less hardware overhead for detecting all non-redundant faults in benchmark circuits than the structures previously proposed.

A synthesis procedure has been described for synthesizing the BIST structures from incompletely specified deterministic test sets. Further research will concentrate on extending this scheme to partial and multiple scan chains.

References

[AgCe81] V. K. Agarwal, E. Cerny: "Store and Generate Built-In Testing Approach", Proc. 11th Int. Symp. Fault-Tolerant Computing, 1981, pp. 35-40

[AkJa89] S. B. Akers, W. Jansz: "Test Set Embedding in Built-In Self-Test Environment", Proc. IEEE Int. Test Conf., 1989, pp. 257-263

[BCR83] Z. Barzilai, D. Coppersmith, A. L. Rosenberg: "Exhaustive Generation of Bit Patterns with Applications to VLSI Self-Testing", IEEE Trans. on Comp., Vol. C-32, No. 2, Feb. 1983, pp. 190-194

[BMS87] P. Bardell, W. H. McAnney, J. Savir: "Built-In Test for VLSI", New York: Wiley-Interscience, 1987

[BRAY84] R. K. Brayton, G. D. Hachtel, C. McMullen, A. Sangiovanni-Vincentelli: "Logic Minimization Algorithms for VLSI Synthesis", Boston: Kluwer Academic Publishers, 1984

[Brgl85] F. Brglez, H. Fujiwara: "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran", Proc. of Int. Symp. on Circuits and Systems, 1985, pp. 663-698

[Brgl89] F. Brglez, D. Bryan, K. Kozminski: "Combinational Profiles of Sequential Benchmark Circuits", Proc. of Int. Symp. on Circuits and Systems, 1989, pp. 1929-1934

[ChPr95] M. Chatterjee, D. K. Pradhan: "A Novel Pattern Generator for Near-Perfect Fault Coverage", 13th IEEE VLSI Test Symposium, 1995, pp. 417-425

[DaMu81] W. Daehn, J. Mucha: "Hardware Test Pattern Generators for Built-In Test", Proc. IEEE Int. Test Conf., 1981, pp. 110-113

DUFA95] C. Dufaza, H. Viallon, C. Chevalier: "BIST Hardware Generator for Mixed Test Scheme", Proc. IEEE European Design and Test Conf., 1995, pp. 424-430

[EiLi83] E. B. Eichelberger, E. Lindbloom: "Random Pattern Coverage Enhancement and Diagnosis for LSSD Logic Self-Test", IBM Journal of Research and Development, Vol. 27, No. 3, May 1983, pp. 265-272

[HELL92] S. Hellebrand, S. Tarnick, J. Rajski, B. Courtois: "Generation of Vector Patterns Through Reseeding of Multiple-Polynomial Linear Feedback Shift Registers", Proc. IEEE Int. Test Conf., 1992, pp. 120-129

[HELL95] S. Hellebrand, B. Reeb, S. Tarnick, H.-J. Wunderlich: "Pattern Generation for a Deterministic BIST Scheme", Proc. Int. Conf. on Computer-Aided Design, 1995, pp. 88-94

[HWH90] S. Hellebrand, H.-J. Wunderlich, O. F. Haberl: "Generating Pseudo-Exhaustive Vectors for External Testing", Proc. IEEE Int. Test Conf., 1990, pp. 670-679

[KMZ79] B. Koenemann, J. Mucha, G. Zwiehoff: "Built-In Logic Block Observation Techniques", Proc. IEEE Int. Test Conf., 1979

[KOEN91] B. Koenemann: "LFSR-Coded Test Patterns for Scan Design", Proc. Europ. Test Conf., Munich 1991, pp. 237-242

[OWM87] M. J. Ohletz, T. W. Williams, J. P. Mucha: "Overhead in Scan and Self-Test Designs", International Test Conference, 1987, pp. 460-470

[SIS92] E. M. Sentovich et al.: "SIS: A System for Sequential Circuit Synthesis", UCB Electronics Research Laboratory, Memorandum No. UCB/ERL M92/41, 1992

[StWu91] A. Ströle, H.-J. Wunderlich: "TESTCHIP: A chip for weighted random pattern generation, evaluation, and test control", IEEE Journal of Solid State Circuits, July 1991, Vol. 26, Number 7, pp. 1056-1063

[ToMc95a] N. A. Touba, E. J. McCluskey: "Transformed Pseudo-Random Patterns for BIST", 13th IEEE VLSI Test Symposium, 1995, pp. 410-416

[ToMc95b] N. A. Touba, E. J. McCluskey: "Synthesis of Mapping Logic for Generating Transformed Pseudo-Random Patterns for BIST", Proc. IEEE Int. Test Conf., 1995, pp. 674-682

[WaMc86] L.-T. Wang, E. J. McCluskey: "Circuits for Pseudo-Exhaustive Test Pattern Generation", Proc. IEEE Int. Test Conf., 1986, pp. 25-37

[WLEF89] J.A. Waicukauski, E. Lindbloom, E.B. Eichelberger, O.P. Forlenza: "A Method for Generating Weighted Random Test Patterns", IBM J. Res. Develop., Vol. 33, No. 2, March 1989, pp. 149-161

[Wu85] H.-J. Wunderlich: "PROTEST: A Tool for Probabilistic Testability Analysis", Proc. IEEE and ACM 22nd Design Automation Conference, 1985, Las Vegas, pp. 204-211

[Wu87] H.-J. Wunderlich: "Self Test Using Unequiprobable Random Patterns", Proc. 17th Int. Symp. Fault-Tolerant Computing, Pittsburgh 1987, pp. 258-263