A Method for Trading Test Time, Area and Fault Coverage in Datapath BIST Synthesis

D. Berthelot, M.L. Flottes, B. Rouzeyre
Laboratoire d'Informatique, de Robotique et de Micro-Electronique de Montpellier,
U.M.R. 5506 CNRS/Université de Montpellier 2, 161 rue Ada, 34392 Montpellier Cedex 5, France

Abstract

This paper presents a method for deriving a BIST specification from the initial specification of datapaths. This method minimizes BIST area overhead under test time constraint while guaranteeing a user chosen fault coverage. The designer can thus explore a wide range of solutions and keep the one that best fits with design constraints. Results show great improvements over lower level techniques.

1. Introduction

The expected benefits of BIST insertion techniques at gate or RT level (e.g. [1, 2, 3, 4, 5, 6, 7]) are limited because the architecture is fixed. After synthesis, test insertion can only degrade the initial design by adding extra test resources and interconnects. That is the reason why many research groups study BIST insertion before architecture definition. In fact, BIST scheme may be decided during HLS (High Level Synthesis) e.g [8] or even before at behavioral level. Several high level BIST insertion methods have been proposed in the literature, they can be broadly classified into two categories: those that target minimal area overhead [9, 10, 11] and those that target minimal number of test sessions [12]. Three remarks can be made concerning these works. First, in any case they are based on specific HLS for BIST algorithms and, consequently, corresponding techniques cannot be used with a classical HLS flow. Second, the FC (Fault Coverage) is evaluated a posteriori, by fault simulation for instance. Consequently, it cannot be used to guide the test insertion process. Finally, related methods suffer from lack of generality in the sense they target a single type of TPGs (Test Vector Generators) and CMPs (response CoMPactors). However, the need to combine several test pattern generation and compaction techniques arises because each technique has its strong and weak points. In one hand, the initial design may be more or less favorable to the intrusion of a given type of test resources in terms of area overhead. On the other hand, pseudo-exhaustive test pattern generators, as arithmetic ones, are well suited for testing regular structures (adders, substracters, multipliers...) while pseudo-random sequences, generated with the help of LFSRs for instance, give better results on random logic. Combining several test resources types has already been proven to be very powerful at RT level [13] as well as behavioral level [14].

This paper describes a new method for implementing a test-per-clock BIST scheme for operators in data-paths. This method targets minimal area overhead *while respecting a test time limit* (TL). It aims at BISTing data paths as soon as possible in the design flow. The FC to achieve on data path's operators is a user-given parameter; it is used to guide the choice of a TPG among all possibilities. Finally, several types of TPGs/CMPs are eventually inserted within the same design for better optimization.

Section 2 discusses our objectives. Section 3 describes the proposed test synthesis methodology. Experimental results are presented in section 4.

2. Test Time and Definitions

While area overhead is the usual main criterion when BISTing a circuit, it had been shown [13] that a light increase in area may lead to large savings on test time. So we state the problem as minimizing area while respecting TL. Using various TL values, the designer can thus explore a wide range of BIST solutions and select the one that best fits with its constraints. Apart from the above mentioned drawbacks, published methods targeting test time minimization suffer from the following limitations:

- All operators are assumed to be tested by the same type of TPG with the same number of test patterns. In other words, these methods minimize the number of test sessions and not really the test time. Test parallelism is limited by test resources conflicts.
- Area overhead is not an input parameter but just a consequence.

Thus these methods do not allow trade-off between area overhead and test time. An exception is the method presented in [15]. In this method datapath is not partitioned into kernels. A single LFSR and a single MISR are connected to the primary I/Os. For a given test length some control/observation points are added to the datapath in order to increase fault coverage. If achieved fault coverage is not sufficient, test length is increased and so on and so forth. This method leads generally to very long test time if a high FC is desired because of the use of a single pseudo-random TPG and the highly sequential nature of the datapaths.

Here we do not assume such restrictions. Test time depends on both the kinds of TPG used for each operator and on scheduling of test sessions.

TPG	Module	FC	#Vect	Type	Area
TPG1	RC-Adder	100.0	15	Arith based	-
TPG2	RC-Adder	100.0	15	LFSR based	10
TPG3	RC-Adder	100.0	17	LFSR based	20
TPG4	CS-Multiplier	100.0	214	LFSR based	15
TPG5	RandomLog1	95.2	1200	LFSR	5
TPG5	RandomLog1	90.4	255	LFSR	5

Fig. 1: TPG's Test Library sample

Test resource characteristics are stored in a test library. For each resource, the library includes a functional model and the logic implementation when necessary (e.g. feedback loop and shift operation of an LFSR). Concerning the TPGs, the test library also includes the corresponding achievable FC for a given test length. Fig.1 gives some examples. From this table we can see that TPG1, an arithmetic TPG, achieves 100% of FC on RC-Adder, a Ripple-Carry adder, with the help of 15 test vectors. TPG5 is an other example. It achieves respectively 90,4% and 95,2% of FC on operator RandomLogic, if the test length is extended from 255 to 1200 vectors. FCs and test lengths are computed in a pre-process task for every pair (TPG, operator). For each entry in this table, area corresponding to the modification of registers in test resources is also stored. Only entries in the library for which the desired fault coverage can be achieved are considered in the following.

We use the following notations and definitions.

TPG(M_i) and CMP(M_i) denotes the type of TPG and CMP used to test operator M_i. Test time necessary to test M_i with TPG(M_i) is denoted Time (M_i, TPG(M_i)).

Definition 1: A test session is a set of operators tested concurrently. In order to simplify the BIST controller, we assume that in a session the tests of concerned modules start and end at the same time. TS is the set of test sessions.

Definition 2: The duration T of a test session S is the test time of the longest-test-time operator in the session, that is T(S) = Max {Time(M_i,TPG(M_i)).

Definition 3: The total time TT for testing operators grouped into test sessions is given by : $TT = \sum_{S \in TS} T(S)$

The organization of individual tests into test sessions is constrained by the following properties.

Property 1: A TPG can be used for testing several operators in a test session or in different sessions.

Property 2: Several operators can not share a CMP in one test session. Conversely, a CMP can be used for several operators in different sessions.

Theorem: The number of ways of grouping the tests of n operators into test sessions is given by: $P_n = \sum_{j=0}^{n-1} C_{n-1}^{j} P_j$

with $C_n^p = \frac{n!}{(n-p)!\,p!}$ (P_n are known as the Stirling coefficients). P_n growths very quickly with n (e.g. P_7 = 876)

3. Test Synthesis

BIST definition is performed at behavioral level. It consists in deriving a behavioral description for the test mode from the original specification of the design. Both TBD (Test Behavioral Description) and IBD (Initial Behavioral description) are then jointly synthesized using a classical high level synthesis flow for full optimization of the entire architectural solution.

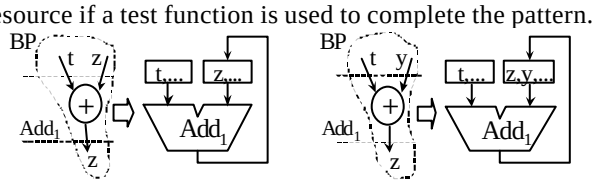
The IBD is first scheduled under timing and hardware constraints, then operators are allocated and assigned to operations. The actual operators to test are thus determined. The so-computed intermediate level of representation, SDFG (Scheduled Data Flow Graph), and the test library support the TBD definition.

The TBD is built up from fragments extracted from the SDFG, these fragments are called BPs (Behavioral Patterns) in the following. Section 3.1 discusses BPs extraction and section 3.2 describes TBD definition.

3.1 Behavioral Patterns in the SDFG

A BP is a fragment of the original SDFG. It defines the behavioral components involved in a future test resource and the way they are interconnected.

It exists two types of behavioral patterns. A type1-BP is complete in the sense that the pattern is sufficient to lead to the generation of a test resource after register assignation. Conversely, a type-2 BP may lead to the generation of a test resource if a test function is used to complete the pattern.



a: Obligatory type1-BP b: Potential type1-BP

Fig.2: Type-1 Behavioral patterns

Fig.2 gives two examples of type1-BPs; both of them may be used to implement a CMP based on an arithmetic compaction technique. The fragment in Fig.2.a includes an adder performing an operation between two variables t and z. The result of this operation is written back into variable z. The BP in this case is composed of one operator and two variables: (Add_1, t, z). After synthesis, this BP leads to the RTL pattern presented on the right-hand side Fig.2.a. Such a type1-BP is said 'obligatory' since it leads to the generation of an arithmetic compactor whatever the register assignation is. Conversely, the type1-BP (Add_1, t, y, z) presented in Fig.2.b is a 'potential' pattern. In fact, it leads to the generation of an arithmetic compactor if, and only if, variables z and y are stored into the same register. When a test solution is selected for a given operator, register assignations involved by the corresponding BP are stored (e.g. assignations involved by a potential type1-BP).

Fig.3 gives an example of a type2-BP. This BP (x, y) can be

used for generating an LFSR-type TPG if a test function (LF1) is added to complete the pattern (see right-hand side, Fig.3). The area overhead of the test function comes from the test library (col.6 in Fig.1). Note that any pair of variables connected to the operator under consideration in the original SDFG is a candidate type2-BP for this operator. For instance the pairs (x,u), (t,u) and (t,y) are also candidate BPs for the test of the adder.

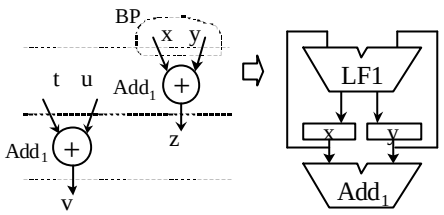


Fig. 3: Type-2 Behavioral pattern

Note that if several operators can be tested by the same type2-BP, only one test generation function (e. g. LF1) is introduced in the data path. All pairs of registers used to store test patterns on concerned operator's inputs are connected to this extra function (see Fig.4).

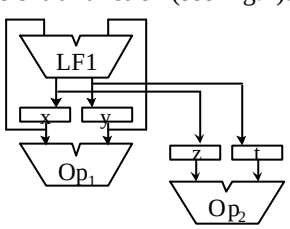


Fig. 4: Type-2 Behavioral pattern

The first task to perform before TBD definition is to build up a set of test solutions for each operator in the data path. These solutions are BPs extracted from the SDFG.

3.2 TBD Definition

The problem to define a test scheme is threefold:

1/ We have to determine the type of test resource used for each operator. This influences the area overhead and the test time.

2/ Test resources have to be instantiated. The number of implemented test resources influences the test time and the area overhead too. For instance two CMPs of the same type may be used to test two operators in parallel, while if only one CMP is instantiated the two operators must be tested in two distinct sessions.

3/ We have to determine the test sessions schedule to respect the user given time limit TL.

The main problem while defining the TBD is to evaluate a very large number of test scheme solutions in order to select the best one. Test schemes need to be evaluated in terms of test time and in terms of area overhead. Test time has already been discussed in section 2. The constraint is TT<TL. Area overhead evaluation is discussed in section a/ below.

In order to find the best solution in an acceptable amount of time, the search process is based on a branch and bound technique. Solutions are ordered before their evaluation in order to speed up the convergence to the best solution and to improve the pruning process. The procedure used to find the best overall solution is detailed in section b/.

a/ Area overhead estimation

As already mentioned above, a large number of solutions need to be evaluated, consequently area overhead must be evaluated with low computational effort. It is thus estimated as the sum of the area overhead introduced by test resources and the area overhead introduced by extra registers.

Area overhead introduced by a given type of test resource is taken from the test library.

Area overhead related to extra registers is the difference between the number of registers needed to implement the whole specification (IBD+TBD) and the number of registers needed to implement the initial specification (IBD). Let N_{Reg} be the number of registers required to implement the IBD. N_{Reg} can be computed by performing a first register assignation on the IBD. Let $Test_{Reg}$ be the minimal number of registers used during the test mode, it can be computed from the number of the number of registers required in every test session as follows. Let $\{Op_k\}$ be the set of operators tested during a test session S, $TPG(Op_k)$ and $CMP(Op_k)$ the test resources involved during Op_k testing, $Reg(TPG(Op_k))$ resp. $Reg(CMP(Op_k))$ be the number of registers necessary to implement $TPG(Op_k)$ resp. $CMP(Op_k)$. $Test_{Reg}$ is given by:

$$Test_{Reg} = \max_{S \in TS}\left(\sum_{Op_k \in S} Reg(TPG(Op_k)) + \sum_{Op_k} Reg(CMP(Op_k)) \right)$$

Since the IBD and TBD lie in exclusive branches, the lifetimes of their variables do not overlap. Thus their variables can share the same registers. The number of registers necessary to implement IBD+TBD is estimated as $\max(Test_{Reg}, N_{Reg})$. The number of extra test registers due to BIST synthesis is taken as:

$$\Delta Test_{Reg} = \max(Test_{Reg}, N_{Reg}) - N_{Reg}$$

b/ Branch and bound

Best solution search process is detailed in Fig.5.

First pruning is performed on TPGs (①). Each operator, Op_k, can be tested with the help of several TPGs, $TPGs(Op_k)=\{TPG(Op_k)_m,\ m=1,...\}$ in the following. $TPGs(Op_k)$ is ordered in such a way that TPGs that can be shared by a maximum of operators are considered first. $Priority1(TPG(Op_k)_m)$ allows ordering $TPGs(Op_k)$:

$$Priority1(TPG(Op_k)_m) = \sum_{Op_i \neq Op_k} \begin{cases} 1 & if \quad TPG(Op_k)_m \in TPGs(Op_i) \\ 0 & otherwise \end{cases}$$

TPGSet represents a sufficient set of test generators for testing all operators Op_k. TPGSets are built up from all combinations of the ordered sets $TPGs(Op_k)$.

The second pruning is performed with respect to the user-given test time threshold TL (②). A scheduling solution, SS, is build up incrementally. At each step, the test of a new operator is added to the preceding partial schedule, partial-SS. If the corresponding test time, Tt(partial-SS), go

```
①For each TPGSet_i
 If Area(TPGSet_i)+Area(?Test_Reg)<BestArea
②For each schedule solution
     If TT<TL
③       For each CMPSet_j
          If Area(TPGSet_i)+ Area(CMPSet_j) +Area(?TestReg)<BestArea
             BestArea= Area(TPGSet_i)+ Area(CMPSet_j) +Area(?TestReg)
             Store TPGSet_i,CMPSet_j
          EndIf
        EndFor
     EndIf
  EndFor
 EndIf
EndFor
```

Fig. 5: Test resource selection under test time constraint

beyond TL, partial-SS is not further explored. Here again, a priority function, Priority2(Op_k), is used to order the operators to schedule. The ones that need the largest number of test patterns are considered first:

$$Priority2(Op_k)=Time(Op_k,TPG(Op_k))$$

The last step is dedicated to the selection of CMPs (③). As for TPG's selection, we have to evaluate several sets of solutions, CMPSets. These sets are build up incrementally, if Area(partial-CMPSet)>BestArea then the process stop for the corresponding set of compactors and a new CMPSet is built up.

At the end of the procedure, test sessions are scheduled and TPGs/CMPs are assigned to operators. The TBD is build up from this information. Extra area due to the BIST intrusion is determined after synthesis of the new behavioral description that includes both TBD and IBD.

4. Results

We applied this method to three classical HLS benchmarks: elliptic wave filter EWFIL, auto-regressive filter ARFIL, and differential equation solver DIFEQ. Each circuit has been synthesized using various area constraints, expressed as number of operators. In all experiments, the desired fault coverage has been fixed to 100%.

The results are presented in the tables below divided into horizontal blocks. Each block compares several BIST strategies for the same area constraints. Column 2 gives the user given test time limit on line 'Initial' (e.g. < 500) and the actual test time for different strategies. Columns 3 to 7 give the number of operators, registers and multiplexor inputs in the designs. Columns 8 and 9 refer to the number of EXOR-based feedback operators used to create LFSR and MISR. The arithmetic TPGs/CMPs, which do not involve area overhead, are not referred in the tables. Signatures are propagated to observation points (primary output or observable registers) via I-paths [16] when possible. Otherwise a shift facility is added to the register storing the signature, this is referred by ShiftReg column.

After BIST insertion, designs are expanded to gate-level using a commercial tool and a 0.7 µ.m library. Areas are measured in terms of number of cell units. Area overhead is given in columns 11 to 14 for 4 datapath bit-widths.

A first set of experiment aims at proving the interest of a high-level test synthesis compared to a RTL technique [13]. A second set of experiments highlights the interest of hybrid test schemes including several types of test resources.

4.1 Comparison with RT approach

In this section, we compare the results of the presented methods (referred as BBIST for behavioral BIST in the tables below) with those obtained using a RT level BIST insertion method (referred as RTBIST). We used our HLS tool (MACH) to generate the RTL descriptions. The line BMinArea refers to designs obtained through the method presented in [14], i.e. a behavioral test synthesis technique targeting minimal area overhead.

It can be seen on these tables that in some case BIST insertion at RT level does not allow satisfying the test time limit. The main reason is the fixed structure of the interconnection network. It can also be seen that our method always leads to the best designs whatever the example and the test time limit. They are generally far better than those obtained at RT level. That's mainly because the RTBIST method must adapt/modify the structure of the circuit in order to parallelize test sessions. Conversely our method generates one (adapted) architecture for each test time constraint.

Finally it can be observed that for a low increase over the minimal area overhead (e.g. 6.37% to compare with 6,51% i.e. an increase of 0.14% in first block in table 1), test time decreases a lot (from 496 to 248).

4.2 Benefit of using various test resources types

Here we applied our method in restricting the set of usable test resources to LFSR and MISR (line LFSR). Then the so synthesized solutions are compared with synthesis solutions where all kinds of TPGs/CMPs are accepted (line all). Using various kinds of resources types always leads to better results than using a single kind of TPGs and CMPs. Area overheads vary by twice as much. One of the reasons is that arithmetic CMP, which do not involve area overhead, can be used instead of MISR. In a counterpart, some designs can not be tested if TPGs/CMPs types are restricted to arithmetic structures.

5 Conclusion

The proposed method is the first one to seek a minimal area BIST solution while respecting a test time constraint. Acting at behavioral level, it lets the high-level synthesis process to care about the optimization of both the initial behavioral description and the BIST behavioral description.

Circuit	Test Time	+	-	*	Reg	Mux	LFSR	MISR	Shift Reg	Area 8 bits	Area 16 bits	Area 32 bits	Area 64 bits
Initial	<300	2	0	2	8	34				Initial Circuit			
RTBIST	248	2	0	2	8	48	2	1	1	9.79%	5.45%	2.87%	1.52%
HLS Bist	248	2	0	2	8	44	1	0	1	6.51%	3.02%	1.70%	0.90%
BMinArea	496	2	0	2	8	42	1	1	0	6.37%	2.90%	1.60%	0.87%
Initial	<300	2	0	3	8	41				Initial Circuit			
RTBIST	-	2	0	3	8	41				No solution at RTL			
HLS Bist	248	2	0	3	8	51	1	1	0	6.64%	3.06%	1.52%	0.79%
Initial	<500	2	0	3	8	41				Initial Circuit			
RTBIST	462	2	0	3	8	55	2	2	1	9.22%	4.85%	2.35%	1.21%
HLS Bist	462	2	0	3	8	54	1	0	1	6.28%	2.85%	1.54%	0.80%
BMinArea	727	2	0	3	7	52	1	0	2	2.29%	0.86%	0.45%	0.23%
Initial	<300	2	0	4	8	48				Initial Circuit			
RTBIST	-	2	0	4	8	48				No solution at RTL			
HLS Bist	231	2	0	4	8	59	1	2	0	7.76%	3.26%	1.52%	0.78%
Initial	<500	2	0	4	8	48				Initial Circuit			
RTBIST	445	2	0	4	8	64	2	2	0	8.23%	4.16%	1.99%	1.02%
HLS Bist	445	2	0	4	8	59	1	1	0	5.79%	2.59%	1.26%	0.65%
BMinArea	958	2	0	4	8	62	1	0	1	4.78%	2.42%	1.28%	0.66%
Initial	<300	2	0	5	8	51				Initial Circuit			
RTBIST	-	2	0	5	8	51				No solution at RTL			
HLS Bist	231	2	0	5	8	64	1	4	0	10.61%	4.11%	1.81%	0.91%
Initial	<500	2	0	5	8	51				Initial Circuit			
RTBIST	462	2	0	5	8	75	4	4	0	11.92%	5.79%	2.66%	1.34%
HLS Bist	462	2	0	5	8	64	1	1	0	5.52%	2.43%	1.18%	0.60%
BMinArea	1189	2	0	5	7	62	1	0	2	3.28%	1.40%	0.73%	0.37%

Table 1: Comparison RTBIST / BBIST (trade-off time/area)ARFIL

Initial	<300	1	1	2	5	19				Initial Circuit			
RTBIST	-	1	1	2	5	19				No solution at RTL			
HLS Bist	253	1	1	2	5	27	1	1	0	11.19%	5.07%	2.50%	1.29%
BMinArea	270	1	1	2	5	25	1	1	0	8.58%	4.41%	2.15%	1.10%
Initial	<300	1	1	3	6	18				Initial Circuit			
RTBIST	-	1	1	3	6	18				No solution at RTL			
HLS Bist	236	1	1	3	6	35	1	2	0	13.89%	5.90%	2.80%	1.42%
Initial	<500	1	1	3	6	18				Initial Circuit			
RTBIST	467	1	1	3	6	40	4	1	1	13.54%	6.36%	3.13%	1.58%
HLS Bist	467	1	1	3	6	40	1	0	1	11.55%	5.20%	2.72%	1.39%
BMinArea	501	1	1	3	6	34	1	1	0	9.59%	4.74%	2.34%	1.19%

Table 2: Comparison RTBIST / BBIST (trade-off time/area)DIFEQ

Results show the interest of directly synthesizing circuits adapted to the test time constraint rather than to work at RT level.

This work is the first one in our knowledge to use various kinds of test resource sin a single design. This feature leads to minimize area overhead for a given test parallelism. Moreover test time of individual MUTs can be optimized by the choice of the TPG type. Presented results show that area overheads are up to 50% less than when BIST insertion is considered at RT level or when a single test scheme is considered.

References

[1] A.P. Stroele, H.-J. Wunderlich, "Hardware-Optimal Test Register Insertion", IEEE Trans. on CAD, pp 531-539, Vol7, No. 6, June 1998.

[2] A. Krasniewski, A. Albicki, "Automatic Design of Exhaustively Self-Testing Chips with BILBO Operators", ITC, pp 362-371, 1985.

[3] C.L. Hudson, G.D. Peterson, "Parallel Self-Test with Pseudo-Random Test Patterns, ITC, pp 954-963, 1987.

[4] P.R. Chalasani et al., "Design of Testable VLSI Circuits with Minimum Area Overhead", IEEE Trans. on Computers, Vol. 38, No.10, Oct. 1989.

[5] A. Basu, T.C. Wilson, D.K. Banerji, J.C. Majithia, "An Approach to Minimize Testability Overhead for BILBO based Buil-In Self-Test, Inter. Conf. on VLSI Design, pp 354-355, 1992.

[6] S-P Lin, C.A. Njinda, M. A. Breuer, "Generating a Family of testable Designs Using the BILBO Methodology", Jetta, N°4, pp 7189, 1993.

[7] D. Gizopoulos, A. Paschalis et Y. Zorian, "An effective BIST scheme for Datapaths". ITC 1996, pp 76-85.

[8] I. Parulkar, S. K. Gupta et M. A. Breuer. "Introducing Redundant Computations in a Behavior for Reducing BIST Resources". DAC, pp. 548-553, 1998.

[9] L. Avra, "Allocation and Assignment in High-Level Synthesis for Self-Testable Data Paths", ITC, pp.463-472, 1991.

[10] H. Harmanani, C. Papachristou, S. Chiu, M. Nourani, "SYNTEST: An Environment for System-Level Design for Test", EDAC, pp. 402-407, 1992.

[11] I. Parulkar, S.K. Gupta, M.A. Breuer, "Allocation Techniques for Reducing BIST Area Overhead of Data Paths", Jetta, Vol. 13, pp 149-166, 1998
[12] I.G. Harris, A. Orailoglu, "Micro-architectural Synthesis of VLSI Designs with High Test Concurrency", 31st DAC, pp206-211, 1994.
[13] D. Berthelot, M.L. Flottes, B. Rouzeyre, "BISTing Datapaths under heterogeneous Test Schemes", Jetta, n°14, pp 115-123, 1999.
[14] D. Berthelot, M.L. Flottes, B. Rouzeyre, "BISTing Data Paths at Behavioral Level" Submitted to ITC 2000.
[15] C. A. Papachristou, M. Baklashov, K. Lai, "High Level Test Synthesis for Behavioral and Structural Design", Jetta, N°13, pp 167-188, 1998.
[16] M.S. Abadir, M.A. Breuer "Constructing optimal test schedules for VLSI circuits having built-in test hardware", 15th Intl Fault Tolerant Computing Conf., pp:165-170, 1985.

Circuit	Test Time	+	-	*	Reg	Mux	LFSR	MISR	Shift Reg	Area 8 bits	Area 16 bits	Area 32 bits	Area 64 bits
Initial	<300	1	0	2	8	18				Initial Circuit			
RTBIST	231	1	0	2	8	34	2	1	1	12.64%	6.47%	3.33%	1.74%
HLS Bist	231	1	0	2	8	27	1	1	0	9.34%	4.17%	2.08%	1.08%
BMinArea	479	1	0	2	8	25	1	1	0	6.88%	3.53%	1.74%	0.90%
Initial	<300	2	0	2	8	28				Initial Circuit			
RTBIST	248	2	0	2	8	40	2	0	1	7.38%	3.77%	2.08%	1.10%
HLS Bist	248	2	0	2	8	38	1	0	1	7.03%	3.07%	1.71%	0.91%
BMinArea	496	2	0	2	8	37	1	0	2	7.03%	3.07%	1.71%	0.91%

Table 3: Comparison RTBIST / BBIST (trade-off time/area)EWFI

Circuit	Test Time	+	-	*	Reg	Mux	LFSR	MISR	Shift Reg	Area 8 bits	Area 16 bits	Area 32 bits	Area 64 bits
Initial	<300	2	0	2	8	34				Initial Circuit			
LFSRs	231	2	0	2	8	45	1	2	0	12.03%	5.65%	2.81%	1.49%
All	248	2	0	2	8	44	1	0	1	6.51%	3.02%	1.70%	0.90%
Initial	<300	2	0	3	8	41				Initial Circuit			
LFSRs	231	2	0	3	8	50	1	3	0	11.06%	4.57%	2.07%	1.07%
All	248	2	0	3	8	51	1	1	0	6.64%	3.07%	1.53%	0.79%
Initial	<500	2	0	3	8	41				Initial Circuit			
LFSRs	445	2	0	3	8	51	1	2	0	9.05%	3.92%	1.86%	0.96%
All	462	2	0	3	8	54	1	0	1	6.28%	2.85%	1.55%	0.81%
Initial	<300	2	0	4	8	48				Initial Circuit			
LFSRs	231	2	0	4	8	56	1	4	0	10.74%	4.11%	1.77%	0.90%
All	231	2	0	4	8	59	1	2	0	7.76%	3.26%	1.52%	0.78%
Initial	<500	2	0	4	8	48				Initial Circuit			
LFSRs	445	2	0	4	8	58	1	2	0	7.44%	3.10%	1.43%	0.73%
All	445	2	0	4	8	59	1	1	0	5.79%	2.59%	1.27%	0.65%
Initial	<800	2	0	4	8	48				Initial Circuit			
LFSRs	445	2	0	4	8	58	1	2	0	7.44%	3.10%	1.43%	0.73%
All	676	2	0	4	8	63	1	0	1	5.81%	2.59%	1.37%	0.70%

Table 4: Comparison of two test libraries (ARFIL)

Circuit	Test Time	+	-	*	Reg	Mux	LFSR	MISR	Shift Reg	Area 8 bits	Area 16 bits	Area 32 bits	Area 64 bits
Initial	<300	1	1	2	5	16				Initial Circuit			
LFSRs	236	1	1	2	5	28	1	2	0	15.63%	6.73%	3.20%	1.64%
All	253	1	1	2	5	27	1	1	0	11.19%	5.08%	2.51%	1.29%
Initial	<300	1	1	3	6	18				Initial Circuit			
LFSRs	236	1	1	3	6	34	1	3	0	16.26%	6.60%	3.03%	1.54%
All	236	1	1	3	6	35	1	2	0	13.89%	5.90%	2.81%	1.43%
Initial	<500	1	1	3	6	18				Initial Circuit			
LFSRs	467	1	1	3	6	39	1	2	9	15.73%	6.83%	3.30%	1.68%
All	467	1	1	3	6	40	1	0	1	11.55%	5.20%	2.72%	1.40%

Table 5: Comparison of two test libraries (DIFEQ)

Circuit	Test Time	+	-	*	Reg	Mux	LFSR	MISR	Shift Reg	Area 8 bits	Area 16 bits	Area 32 bits	Area 64 bits
Initial	<300	1	0	2	8	18				Initial Circuit			
LFSRs	231	1	0	2	8	27	1	2	0	12.93%	5.42%	2.58%	1.34%
All	231	1	0	2	8	27	1	1	0	9.34%	4.17%	2.08%	1.09%
Initial	<300	2	0	2	8	28				Initial Circuit			
LFSRs	231	2	0	2	8	39	1	2	0	12.99%	5.75%	2.84%	1.50%
All	248	2	0	2	8	38	1	0	1	7.03%	3.07%	1.72%	0.91%

Table 6: Comparison of two test libraries (EWFIL)