Design of Scalable Hardware Test Generators for On-Line BIST

Hussain Al-Asaad and John P. Hayes

Advanced Computer Architecture Laboratory
Department of Electrical Engineering and Computer Science
University of Michigan, Ann Arbor, MI 48109-2122

Email: {halasaad, jhayes}@eecs.umich.edu
URL: http://www.eecs.umich.edu/~{halasaad, jhayes}

Brian T. Murray

Electrical and Electronics Department
General Motors R&D Center
Warren, MI 48090-9055

Email: bmurray@gmr.com
URL: http://www.eecs.umich.edu/~lauren

Abstract

This paper briefly reviews on-line built-in self-test (BIST) and shows its importance in concurrent checking. Then a new approach for the design of deterministic BIST hardware test generators is presented. The approach uses high-level models of circuits to identify the classes of tests needed for complete coverage of faults. The test generator is then designed with the following goals: scalability, near-minimal error latency, and complete coverage of the modeled faults. Moreover, the test generators produced are simple and have low hardware overhead. Preliminary case studies of carry-lookahead adders, arithmetic logic units, and barrel shifters show the usefulness of this technique.

1 Introduction

There are four primary parameters to be considered in the design of any on-line testing scheme:

- *Error coverage (EC)*: This is defined as the percentage of modeled errors that are detected, for example, all errors due to single stuck-line (SSL) faults. In safety-critical applications 100% error coverage is mandated.
- *Error latency (EL)*: This is the difference between the first time the error is activated and the first time it is detected. For example, the error latency of duplication with comparison (DWC) [9] is 1 clock cycle. In safety-critical applications, low error latency is required.
- *Hardware redundancy (HR)*: This is the extra hardware needed to perform on-line testing. For example, DWC has about 100% hardware redundancy.
- *Time redundancy (TR)*: This is the extra time needed to perform on-line testing. For example, recomputing with shifted operands (RESO) [11] has about 100% time redundancy.

An ideal on-line testing scheme would have 100% error coverage, error latency of one clock cycle, no hardware redundancy, no time redundancy, require no redesign of the circuit under test (CUT), and impose no functional or structural restrictions on the CUT. Current schemes meet some of these constraints but fail to meet the others. Table 1 evaluates some representative on-line testing techniques based on the four parameters defined above. From this table, we conclude that there is no technique that is applicable to all

Table 1 Comparison of some on-line testing schemes.

Technique	Error coverage	Error latency	Redundancy		Miscellaneous		Target circuits	
			Hardware	Time	Redesign of the CUT	Constraints on the CUT	Combinational	Sequential
Alternating logic [12]	<100	1	0 - 100	> 100	Yes	None	Yes	Yes
RESO [11]	100	1	0 - 100	> 100	No	Regular	Yes	No
RESWO[7]	100	1	0 - 100	0 - 100	No	Regular	Yes	No
REDWC[10]	100	1	0 - 100	0 - 100	No	Regular	Yes	No
DWC [9]	100	1	>100	~0	No	None	Yes	Yes
CBIST[14]	100	$>10^6$	5 -100	~0	No	None	Yes	No
Self monitoring [13]	<100	1	5 - 100	~0	Yes	None	No	Yes

combinational circuits and that guarantees full error coverage, low error latency, and low time and hardware overhead.

On-line BIST is usually implemented with the goals of complete error coverage, near-minimal error latency, and moderate hardware and time overhead. The key idea is to incorporate into the CUT a testing process that guarantees detection of all the target faults within a given time. The first requirement is a complete test for the fault model under consideration. In other words, if the CUT is faulty, then it must produce an erroneous output when the test set (or test sequence in the case of sequential circuits) is applied. The full test set or sequence is not usually applied in consecutive clock cycles. Instead, it is partitioned and applied in a periodic fashion. This implies that the error latency is directly proportional to the test set size. Hence, a near-minimal test set is desirable for on-line BIST. When the CUT is scalable, such as a datapath circuit, it is more practical to design scalable BIST components including the test generator. This makes it possible to develop cell-based designs with embedded BIST.

We first review previous work in designing test generators in Section 2, and then we present our approach for the design of scalable test generators for on-line BIST in Section 3. We conclude and present directions for further research in Section 4.

2 Hardware test generators

The problem of designing deterministic test generators

1. This research was supported by General Motors R&D Center.

for on-line BIST in (combinational) circuits can be described as follows: Given a combinational circuit *C* and a fault model *F*, determine a near-minimal circuit TG to generate a near-minimal test set *S*, i.e., one with near-minimal error latency, that is complete for *F*. One solution to this design problem is to use an exhaustive test set for *S*, for which the test generator can simply be a counter. However, the resulting test application time and error latency are unacceptable if the number of inputs to the circuit under test is more than about 30. Other proposed solutions for TG design use linear feedback shift registers [6]. Like counters, the test sets needed and the error latency may be too large to achieve complete fault coverage.

A simplified version of the design problem of deterministic test generators can be described as follows: Given an unordered deterministic test set *T*, determine a near-minimal circuit TG that will generate a small test set *S* that contains *T* as a subset. Solutions proposed for this simpler problem include cellular automata [2] and non-linear feedback shift registers [4]. A major characteristic of all of the above approaches is that in addition to some combinational logic, TG needs *n* flip-flops to generate tests for an *n*-input CUT. This may increase the area used by the TG to unacceptable levels.

A different solution for the simplified design problem is to synthesize a finite state machine TG that will produce precisely the test set required. The advantage of this technique is that only $\lceil \log |T| \rceil$ flip-flops are needed for a given test set *T*. The disadvantage is that most logic synthesis programs fail to design TG if the number of tests in *T* and the test width are large.

A similar approach is to use a counter followed by a combinational logic [5] to generate the tests as shown in Figure 1. The combinational logic can be implemented using ROMs, PLAs, or basic logic gates. An XOR-network was proposed in [1] to implement the combinational logic of Figure 1. The main design task is to embed a set of deterministic tests into the XOR-network via mapping a subset of the binary counter states into the required deterministic tests. This technique has the advantage of using linear transformations so that linear algebra can be used to reduce the size of the XOR-network. However, by forcing the combinational logic to be linear, the number of deterministic tests that can be embedded becomes severely limited, and hence high fault coverage cannot be guaranteed. Moreover, the hardware overhead of the XOR-network tends to be large.

Yet another way to design deterministic test generators is proposed in [3], where the combinational logic of Figure 1 is limited to inverters and wires. This method identifies compatible circuit inputs that can be connected to the same counter output: inputs are connected together if no redundancy is introduced. Moreover, inputs can be connected via an inverter in a similar way. The advantages of this method are its low hardware overhead and the high speed of the combinational logic. However, it has two problems. First, identifying the inputs to be shorted is equivalent to the problem of redundancy identification, which is not easy to solve. Second, if a small number of inputs are shorted, then the counter size becomes large and the test set tends to be exhaustive, which leads to unacceptable error latency.

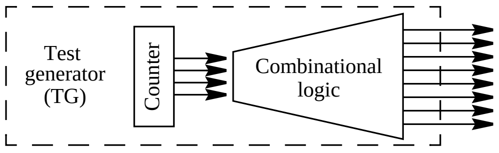

Figure 1 Counter-based test generator.

A deterministic test generator for a bit-sliced CUT, such as a ripple carry adder, can be easily scaled if the size of CUT is increased. However, no proposed technique for the design of deterministic test generators shows how to systematically redesign the test generator if the size of the (non-bit-sliced) CUT is increased. For example, given a test generator for, say, a 4-bit carry-lookahead adder, we cannot infer a similar design for a 32-bit adder. We next present a method for the design of scalable hardware test generators for on-line BIST.

3 Scalable test generators

To systematically scale hardware test generators for on-line BIST, high-level information about the CUT is useful. It allows us to rapidly explore the large number of possible test sets and determine an implementation that has minimum hardware. Another advantage of this approach is the ability to automatically synthesize the test generator with the CUT without extensive analysis.

The test generator TG is designed as follows:

1. Determine a high-level model of the scalable circuit.
2. Use the high-level model to derive a high-level representation for the complete test sets for the modules in the circuit model. Use don't cares in the high-level representation of the test sets as far as possible to simplify the next step.
3. Use high-level propagation and justification to select a specific test set for the overall high-level circuit model. The test set should be regular so that it can be easily scaled as well as generated. Regularity in the test set can be spatial or temporal, for example:
   a. Test t_{i+1} is a shifted version of test t_i.
   b. Test t_j is the bitwise complement of test t_i.
   c. There are repeated sub-patterns in test t_i.
4. Design the test generator based on the determined regular and scalable test set.

We illustrated the proposed design method with several small case studies. Our first case study is the 74283 4-bit carry lookahead adder [15]. A high-level model of the adder was derived in [8] and is shown in Figure 2. Module M_1 produces the generate and propagate signals for the carry lookahead generator module M_2. As shown in [8], tests for M_2 are relatively easy to obtain using M_2's functional description. The testing requirements for M_2 are traced back to M_1 and the resulting test sets are shown in Table 2. This is a condensed high-level representation of M_2's testing requirements, which specifies implicitly all possible sets of 10 tests (the minimum number) that cover all SSL faults in M_2. For module M_1, each pair of bits A_iB_i must be exhaustively tested. The tests for M_2 guarantee the application of 00 and

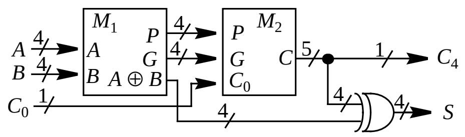

Figure 2 High-level model of the 74283 carry-lookahead adder [8].

11 on each A_iB_i of M_1. Therefore, the remaining requirement for testing M_1 is to apply 01 and 10 to each A_iB_i. The XOR gate is automatically covered by the testing requirements of M_1 and M_2. From this analysis and Table 2 we can show that a complete and minimal test set for an *n*-bit carry-lookahead adder is of size $2n + 2$. One such test set for the 74283 adder of size 10 is shown in Table 3. The ten tests have been selected heuristically to exhibit as much regularity as possible. Note, for example, that the first five tests are the bitwise complements of the second five. Furthermore, certain patterns (shaded in Table 3) are shifted in regular fashion from right to left.

A possible implementation of a test generator for the 74283 adder using the tests of Table 3 is shown in Figure 3. The test generator is based on a ring counter and is easily expandable to any carry-lookahead adder. For an *n*-bit carry-lookahead adder, the test generator requires *n* flip-flops and *n* cells of the linear array. The hardware overhead of the test generator is shown in Table 4. This overhead decreases as the number of bits of the carry-lookahead adder increases.

Our second case study is the 74181 4-bit arithmetic logic unit (ALU) [15]. The high-level model of the ALU in [8] was used to derive a complete test set of size $4n + 8$ for SSL faults with both spatial and temporal regularity. A possible implementation of the test generator for the 74181 is shown in Figure 4. The test generator is based on a ring counter and its derivation is similar to that of the 74283. For an *n*-bit ALU, the test generator requires only *n* flip-flops with some constant logic independent of *n*. The hardware overhead of the test generator is shown in Table 4. This overhead decreases as the number of inputs of the ALU increases. For

Table 2 High-level representation of complete test sets for SSL faults in the 74283 module M_2.

Test #	Selection	A3 B3	A2 B2	A1 B1	A0 B0	C0
1	---	$\oplus$[a]	$\oplus$	$\oplus$	$\oplus$	1
2	---	$\oplus$	$\oplus$	$\oplus$	$\oplus$	0
3	---	$\oplus$	$\oplus$	$\oplus$	0 0	1
4a	One[b]	$\oplus$	$\oplus$	0 0	1 1	x
4b		$\oplus$	$\oplus$	0 0	$\oplus$	1
5a	One	$\oplus$	0 0	1 1	x x	x
5b		$\oplus$	0 0	$\oplus$	1 1	x
5c		$\oplus$	0 0	$\oplus$	$\oplus$	1
6a	One	0 0	1 1	x x	x x	x
6b		0 0	$\oplus$	1 1	x x	x
6c		0 0	$\oplus$	$\oplus$	1 1	x
6d		0 0	$\oplus$	$\oplus$	$\oplus$	1
7	---	$\oplus$	$\oplus$	$\oplus$	1 1	0
8a	All[c]	$\oplus$	$\oplus$	1 1	0 0	x
8b		$\oplus$	$\oplus$	1 1	$\otimes$[d]	0
9a	All	$\oplus$	1 1	0 0	x x	x
9b		$\oplus$	1 1	$\otimes$	0 0	x
9c		$\oplus$	1 1	$\otimes$	$\otimes$	0
10a	All	1 1	0 0	x x	x x	x
10b		1 1	$\otimes$	0 0	x x	x
10c		1 1	$\otimes$	$\otimes$	0 0	x
10d		1 1	$\otimes$	$\otimes$	$\otimes$	0

a. Only one of the inputs is 1.
b. It is sufficient to include one of these tests.
c. All tests must be included, however, they may be compressed in one.
d. At least one of the inputs is 0.

Table 3 A complete, regular, and minimal test set for SSL faults in the 74283.

Test #	A3 B3	A2 B2	A1 B1	A0 B0	C0
1	1 0	1 0	1 0	1 0	1
2	1 0	1 0	1 0	0 0	1
3	1 0	1 0	0 0	1 1	1
4	1 0	0 0	1 1	1 1	1
5	0 0	1 1	1 1	1 1	1
6	0 1	0 1	0 1	0 1	0
7	0 1	0 1	0 1	1 1	0
8	0 1	0 1	1 1	0 0	0
9	0 1	1 1	0 0	0 0	0
10	1 1	0 0	0 0	0 0	0

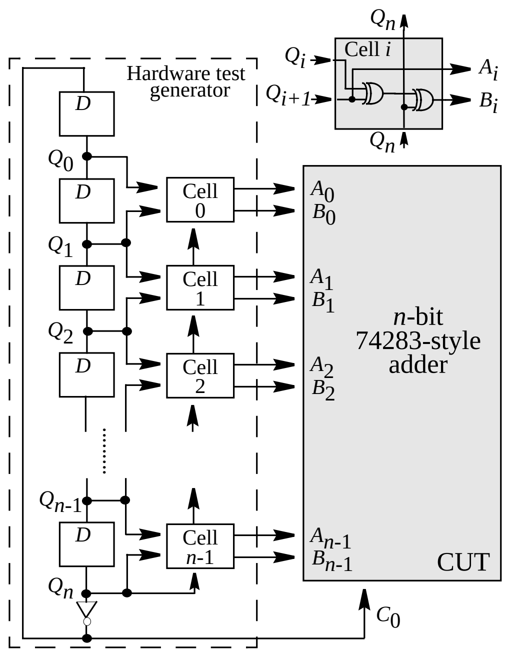

Figure 3 Low-cost scalable hardware test generator for a 74283-style adder.

Table 4 Hardware overhead for the test generators for the circuits studied.

Circuit	*n* = 4	*n* = 8	*n* = 16	*n* = 32	*n* = 48	*n* = 64
74283-like *n*-bit adder using 4-bit CLA	45.5	40.1	36.9	35.8	35.4	35.1
74181-like *n*-bit ALU using 4-bit CLA	23.2	16.1	12.9	11.4	10.9	10.6
74181-like *n*-bit RALU using 4-bit CLA	18.8	12.1	9	7.5	7	6.7
n-bit barrel shifter	70.0	41.5	26.5	18.3	15.4	13.6

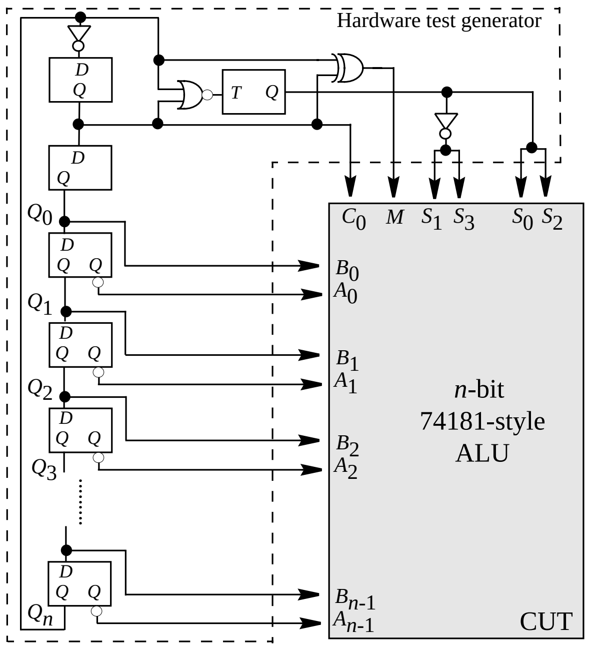

Figure 4 Low-cost scalable hardware test generator for a 74181-style ALU.

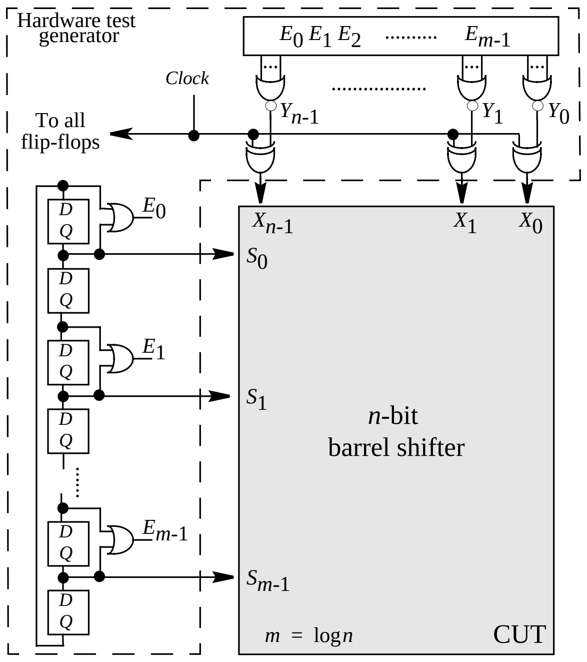

Figure 5 Low-cost scalable hardware test generator for an *n*-bit barrel shifter.

example, it is only 10.6% in the 64-bit case.

In many applications, flip-flops are already available at the inputs of the circuit. An example of this case is a registered ALU (RALU), which appear in many digital signal processing circuits. The test generator shown in Figure 4 can use the available flip-flops to further reduce the area overhead shown in Table 4.

Our third case study is an *n*-bit barrel shifter that implements a *k*-bit (left) shift for $0 \le k \le n-1$. The high-level model of the shifter is composed of $\log n$ *n*-bit multiplexers. A regular test set of size $4 \times \log n$ was derived that aim at testing each stage of the shifter using four tests only. A possible implementation of the resulting test generator is shown in Figure 5. The hardware overhead of the test generator is shown in Table 4. It decreases as *n* increases and amounts to 13.6% for a 64-bit shifter.

4 Conclusions

We have presented an approach for the design of scalable hardware test generators for on-line BIST, and illustrated it for several practical datapath circuits. The resulting test generators produce complete and near-minimal test sets, and so have low error latency. They also have fairly low hardware overhead and are easily expandable to test larger versions of the same target circuits.

Our future research will address other useful datapath circuits of a typical microprocessor such as multipliers and floating-point arithmetic circuits. We hope to develop the approach presented here into a systematic one that can eventually be automated.

References

[1] S. B. Akers and W. Jansz, "Test set embedding in a built-in self-test environment", *Proc. International Test Conference*, 1989, pp. 257-263.

[2] S. Boubezari and B. Kaminska, "A deterministic built-in self-test generator based on cellular automata structures", *IEEE Transactions on Computers*, Vol. 44, pp. 805-816, June 1995.

[3] C. -A. Chen and S. K. Gupta, "A methodology to design efficient BIST test pattern generators", *Proc. International Test Conference*, 1995, pp. 814-823.

[4] W. Daehn and J. Mucha, "Hardware test pattern generation for built-in testing", *IEEE Test Conference*, 1981, pp. 110-113.

[5] R. Dandapani, J. H. Patel, and J. A. Abraham, "Design of test pattern generator for built-in self-test", *Proc. International Test Conference*, 1984, pp. 315-319.

[6] C. Dufaza and G. Cambon, "LFSR based deterministic and pseudo-random test pattern generator structures", *Proc. European Test Conference*, 1991, pp. 27-34.

[7] H. Hana and B. W. Johnson, "Concurrent error detection in VLSI circuits using time redundancy", *Proc. IEEE Southeastcon*, 1986, pp. 208-212.

[8] M. C. Hansen and J. P. Hayes, "High-level test generation using physically-induced faults", *Proc. VLSI Test Symposium*, 1995, pp. 20-28.

[9] B. W. Johnson, *Design and Analysis of Fault Tolerant Digital Systems*, Addison-Wesley, Reading, Mass., 1989.

[10] B. W. Johnson, J. Aylor, and H. Hana, "Efficient use of time and hardware redundancy for concurrent error detection in a 32-bit VLSI adder", *IEEE Journal of Solid-State Circuits*, Vol. 23, pp. 208-215, February 1988.

[11] J. H. Patel and L. Y. Fung, "Concurrent error detection in ALU's by recomputing with shifted operands", *IEEE Transactions on Computers*, Vol. C-31, pp. 417-422, July 1982.

[12] D. Reynolds and G. Metze, "Fault detection capabilities of alternating logic", *IEEE Transactions on Computers*, Vol. C-27, pp. 157-162, December 1978.

[13] S. H. Robinson and J. P. Shen, "Direct methods for synthesis of self-monitoring state machines", *SRC Pub C92074*, 1992.

[14] K. K. Saluja, R. Sharma, and C. R. Kime, "A concurrent testing technique for digital circuits", *IEEE Transactions on CAD*, Vol. 7, pp. 1250-1259, December 1988.

[15] Texas Instruments, *The TTL Logic Data Book*, Dallas, 1988.