On Output Response Compression in the Presence of Unknown Output Values

Irith Pomeranz[1], School of ECE, Purdue University, W. Lafayette, IN 47907, pomeranz@ecn.purdue.edu
Sandip Kundu, Intel Corp., Austin, TX 78746, sandip.kundu@intel.com
Sudhakar M. Reddy[2], ECE Dept., University of Iowa, Iowa City, IA 52242, reddy@engineering.uiowa.edu

Abstract

A circuit may produce unknown output values during simulation of an input sequence due to an unknown initial state or due to the existence of tri-state elements. For circuits tested using BIST, unknown output values make it impossible to determine a single unique signature for the fault free circuit. To accommodate unknown output values in a BIST scheme, we describe a procedure for synthesizing a minimal logic block that replaces unknown output values by a known constant. The proposed procedure ensures that the BIST scheme will be able to detect all the faults detectable by the input sequence applied to the circuit while allowing a single unique signature to be obtained.

1. Introduction

Built-in self-test (*BIST*) [1]-[4] is becoming the accepted solution for at-speed testing of high-performance circuits [5]. Typical *BIST* circuitry is shown in Figure 1 (excluding the logic in the dashed box). Test vectors for the circuit-under-test (*CUT*) are produced by the test-pattern generator (*TPG*). The output response analyzer (*ORA*) compresses the output response of the *CUT* and produces a signature which can be used to determine whether the circuit is fault free or faulty.

One of the problems encountered in output response analysis for complex circuits (e.g., circuits containing buses and tri-state elements) is that the output values produced by the circuit under the sequence generated by the *TPG* may be unknown at certain time units and on certain outputs. Even when deterministic test generation is carried out, it is impossible to characterize and avoid all the situations under which, e.g., bus contention will occur and unknown output values will be produced. Under a test sequence produced by a *TPG*, bus contention is even more difficult to avoid. When it occurs, it may result in unpredictable output values. If the output sequence of the fault free circuit contains unknown values, a single predictable signature cannot be computed. One way to resolve this problem is to compute several signatures, one for each specification of the unknown values. However, this becomes impractical if the number of unknown values is large. A more practical solution is to ensure that the *ORA* does not compress the value of an output at a time unit when it is unknown. *BIST* circuitry appropriate for this case is shown in Figure 1. The block called *CMP* (which stands for *compress*) determines which output values at which time units will be compressed by the *ORA*, and which values will be ignored. Ignored values are replaced by the *CMP* block with a constant (0 in Figure 1). Specifically, the *CMP* output determines whether the multiplexer (*MUX*) output that drives the *ORA* will come from the *CUT* or will be replaced by the constant 0. This ensures that the *ORA* design does not have to be changed, and the same *ORA* used without the logic in the dashed box of Figure 1 manipulates only specified, i.e., {0,1} values, in Figure 1. As a result, a single unique fault free signature can be computed for the circuit. We point out that other variations on the *CMP* and *ORA* blocks are possible.

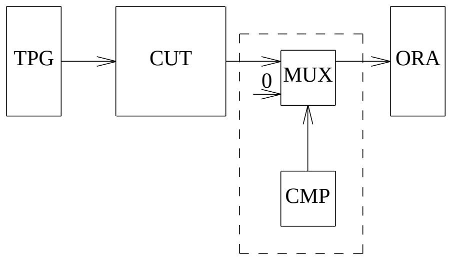


Figure 1: ***BIST*** **circuitry with unknown output values**

When the *CUT* of Figure 1 is a multi-output circuit, there is a multiplexer on every *CUT* output, and the *CMP* block produces a separate control line for every multiplexer.

In this work, we consider the synthesis of a minimal-size *CMP* block in the case of a multi-output synchronous sequential *CUT* whose output sequence contains unknown values. We assume that the *TPG* produces a single input sequence which is used for testing the *CUT*, and that the *CUT* in response produces an output sequence that contains unknown values at specific time units and on specific outputs. The *CMP* block is suitable only for this output sequence. If the *TPG* is changed such that the locations of the unknown values in the output sequence change, the *CMP* block must be redesigned to match the new output sequence produced by the *CUT*. Although we describe the *CMP* synthesis procedure for the case of non-scan circuits, the proposed synthesis procedure can also handle scan circuits with single or multiple scan chains.

———————————

1. Research supported in part by NSF Grant No. CCR-0098091 and in part by SRC Grant No. 2001-TJ-950.
2. Research supported in part by NSF Grant No. CCR-0097905 and in part by SRC Grant No. 2001-TJ-949.

A problem similar to the one addressed here is considered in [6]. However, no details are available in [6] to allow a comparison with the method proposed here.

The paper is organized as follows. In Section 2 we formulate the problem of minimizing the size of the CMP block. In Section 3 we describe a procedure for synthesizing a CMP block based on the formulation of Section 2. We present experimental results in Section 4. Section 5 concludes the paper.

2. Problem formulation

We use the following notation. The output sequence produced by the fault free circuit in response to the TPG sequence is denoted by Z. The value of output i at time unit u obtained under the TPG sequence is denoted by $Z[u,i]$. For a circuit with a test sequence of length L and O primary outputs, Z is an $L \times O$ array. An output value $Z[u,i]$ can be any one of the three values $\{0,1,U\}$ where U stands for an unknown value.

We define an $L \times O$ array W_0 that indicates which output values at which time units are known and may be compressed by the ORA, and which values must not be compressed (i.e., they must be replaced by a constant) since they are unknown. The array W_0 is defined as follows. If $Z[u,i] = U$, $W_0[u,i] = 0$ to indicate that the value of output i at time unit u must not be compressed. If $Z[u,i] = 0$ or 1, $W_0[u,i] = x$ to indicate that the value of output i at time unit u may be compressed.

We also define an array $W_{\max}$ as follows. If $Z[u,i] = U$, $W_{\max}[u,i] = 0$ to indicate that the value of output i at time unit u must not be compressed. If $Z[u,i] = 0$ or 1, $W_{\max}[u,i] = 1$ to indicate that the value of output i at time unit u is compressed. According to $W_{\max}$, every known output value is compressed.

Our goal is to find an $L \times O$ array W with $\{0,1\}$ entries based on which the CMP block shown in Figure 1 will be synthesized. We refer to a value $Z[u,i]$ such that $W[u,i] = 1$ (i.e., such that $Z[u,i]$ will be compressed) as an *observed* value, and we refer to a value $Z[u,i]$ such that $W[u,i] = 0$ as an *unobserved* value. We thus imply that if the effect of a fault f propagates to output i at time unit u and $Z[u,i]$ is compressed, f will be detected. We point out that detection of f is not guaranteed in this case since aliasing may occur and mask the fault effects. However, since aliasing is rare and methods to avoid it exist, the assumption that f will be detected is justified.

The array W that indicates which CUT output values will be compressed and which values will not be compressed must satisfy the following conditions.

(1) Observing the output values corresponding to pairs (u,i) where $W[u,i] = 1$ is sufficient to detect all the circuit faults that are detectable by the TPG sequence. In other words, the observed values must be such that for every detectable fault f, the effect of f propagates to at least one output i at a time unit u where $W[u,i] = 1$.

(2) $W[u,i] \le W_{\max}[u,i]$ for every u and i. This condition guarantees that if $W_{\max}[u,i] = 0$, $W[u,i]$ will be 0 and the corresponding entry will not be compressed. For the remaining time units and outputs, $W_{\max}[u,i] = 1$ and both 0 and 1 values can be used in W.

(3) The CMP logic corresponding to W is minimal. We discuss next a specific implementation of the CMP logic based on a given array W, and the requirements for minimizing it.

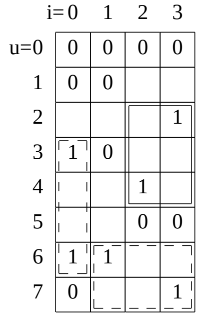
An example of an array W for a circuit with $O = 4$ outputs and a TPG sequence of length $L = 8$ is shown in Figure 2. In this array, $W[u,i] = 0$ if an entry must not be compressed because $Z[u,i]$ is unknown, and $W[u,i] = 1$ if the value of output i at time unit u must be observed to detect a target fault. The empty boxes indicate that $W[u,i] = x$. These values are don't-cares and they correspond to output values that may be compressed or not to minimize the size of the CMP logic.

We consider W as an incompletely specified truth table with inputs representing the L time units $u = 0,1, \cdots ,L-1$ and O outputs corresponding to the CUT outputs. To keep track of the time unit u the CMP block contains a counter, which is reset to 0 when the TPG is initialized, and incremented using the CUT (and TPG) clock.

It is possible to use a general-purpose logic synthesis procedure to find a minimal implementation of the CMP block. However, for practical circuits, the TPG sequence length L and the number of outputs O are typically large, and general-purpose logic synthesis procedures may be inefficient for such problem sizes. Instead, we use the following approach. We construct the CMP block from the basic blocks introduced in [7] and referred to as *comparison unit*s. The input to a comparison unit is the time unit u. Using the decimal form of u, a comparison unit produces the output value 1 if and only if $LB \le u \le UB$, where LB and UB are constants specific to each comparison unit. The maximum number of two-input gates in a comparison unit with m inputs is $2m-1$. The number of gates can be significantly smaller, depending on LB and UB.

To use comparison units for implementing the CMP block, we partition the array W into rectangles of the same logic value. Each rectangle will be implemented by a single comparison unit. For example, consider the rectangle marked with solid lines in Figure 2, which is defined by $2 \le u \le 4$ and $2 \le i \le 3$ and contains the 1's for $(u,i) = (2,3)$ and $(4,2)$. Since this rectangle does not contain any 0's, we can use a comparison unit with $LB = 2$ and $UB = 4$ to implement it. This comparison unit will be used as part of the implementation of the CMP outputs driving the multiplexers on CUT outputs 2 and 3. It will ensure that the value of output 3 at time unit 2 and the value of output 2 at time unit 4 are compressed by the ORA to detect target faults. Additional values will be compressed, belonging to outputs 2 and 3 at time units 2, 3 and 4. Although these values are not necessary for detecting target faults, observing them will increase the ability to detect unmodeled defects.

In general, our goal is to find a minimal number of rectangles to cover all the entries (u,i) such that $W[u,i] = 1$. Each rectangle will be implemented using a single comparison unit. All the comparison units for a given CUT output will then be ORed to form the corresponding output of the CMP. For the example of Figure 2, we use the rectangles shown in the figure. The CMP implementation in this case is shown in Figure 3.



	i=0	1	2	3
u=0	0	0	0	0
1	0	0		
2				1
3	1	0		
4			1	
5			0	0
6	1	1		
7	0			1

Figure 2: Rectangles for Example 1

Instead of rectangles to cover the 1 values in W, it is possible to find rectangles to cover the 0 values in W. With a CMP block implementing a cover of the 0 values, a CUT output value should be replaced by a constant value when the corresponding

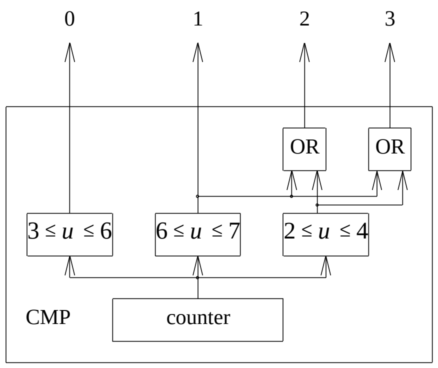


Figure 3: A *CMP* block for Example 1

output of the *CMP* block is 1. We use the cover that results in the smaller number of comparison units. We point out that implementing the *CMP* block based on rectangles that cover W simplies the synthesis procedure but may result in a comparison unit based implementation which is larger than necessary.

3. Synthesis procedure for *CMP* blocks

Synthesis of the *CMP* block consists of two tasks. (1) Determining the time units and outputs where the *CUT* output will be compressed (or observed) to ensure that all the faults can be detected while avoiding unknown output values. This task is equivalent to finding an array W that satisfies the conditions stated in Section 2. (2) Using W to find a minimal number of rectangles for a comparison unit based implementation of the *CMP* block. As part of the proposed synthesis procedure, we carry out these tasks simultaneously to ensure that the smallest possible number of rectangles (and hence comparison units) will be selected. We describe the synthesis procedure in the following subsections.

We already defined in Section 2 the array W_0, which provides information about the necessary 0's in any array W that satisfies the conditions of Section 2. In Subsection 3.1 we define entries that *must* be 1 in W to ensure that all the target faults (the faults detectable by the *TPG* sequence) are detected. These entries are defined based on faults that can only be detected by the *TPG* sequence at a single time unit, on a single output. In Subsection 3.2 we describe a procedure that, given an array W with $\{0,1,x\}$ entries, finds a minimal number of rectangles to cover its 0 (or 1) values. This procedure is used in Subsection 3.3 to guide the selection of additional 1 values in W to ensure that *all* the target faults can be detected by the observed values of W. In Subsection 3.4, this procedure is used for selecting the final set of rectangles that will define the implementation of the *CMP* block.

3.1. Necessary 1's in W

Consider a fault f, which is detected by the *TPG* sequence on a single primary output i at a single time unit u. To ensure that f is detected after output response analysis, we must set $W[u,i] = 1$. We find faults such as f by an n-detection ($n > 1$) fault simulation process. This process drops a fault f from simulation only after f has been detected n times, on n different combinations of outputs and time units. We point out that to identify entries that must be 1 in W, it is sufficient to use $n = 2$. However, in order to collect information that will be useful in Subsection 3.3, we prefer to use a larger value of n.

For every fault f, we find a set $det(f)$, which is the set of time units and primary outputs (u,i) such that f is detected by the *TPG* sequence at time unit u on output i. We then set $W[u,i] = 1$ as follows. Initially, $W = W_0$ to ensure that all the output values that must be ignored are set to 0. For every fault f, if $det(f)$ contains a single entry (u,i), we set $W[u,i] = 1$ to ensure that f can be detected after output response analysis. We refer to this procedure as Procedure 1.

3.2. Finding rectangles for a given array W

In this subsection we describe a procedure that, given an array W with $\{0,1,x\}$ entries, finds a minimal number of rectangles to cover its α values, where $\alpha \in \{0,1\}$. When covering the α values with rectangles, no rectangle must contain the value $\bar{\alpha}$. This procedure will be used in Subsections 3.3 and 3.4.

In the proposed procedure, a rectangle R_j is defined by its time unit and output ranges, $R_j = [u_{sj},u_{ej}]\times[i_{sj},i_{ej}]$. The proposed procedure initially includes every pair (u,i) such that $W[u,i] = \alpha$ in a separate rectangle. Thus, if $W[u,i] = \alpha$, we define a rectangle $R = [u,u]\times[i,i]$. Starting from this initial set of rectangles, the procedure *combines* as many pairs of rectangles as possible. The combination of two rectangles $R_1 = [u_{s1},u_{e1}]\times[i_{s1},i_{e1}]$ and $R_2 = [u_{s2},u_{e2}]\times[i_{s2},i_{e2}]$ results in the rectangle $R_3 = [u_{s3},u_{e3}]\times[i_{s3},i_{e3}]$ where $u_{s3} = \min\{u_{s1},u_{s2}\}$, $u_{e3} = \max\{u_{e1},u_{e2}\}$, $i_{s3} = \min\{i_{s1},i_{s2}\}$ and $i_{e3} = \max\{i_{e1},i_{e2}\}$. The combination of R_1 and R_2 is accepted if R_3 does not contain any $\bar{\alpha}$ value of W, i.e., if there does not exist a pair (u,i) such that $W[u,i] = \bar{\alpha}$, $u_{s3} \le u \le u_{e3}$ and $i_{s3} \le i \le i_{e3}$.

If R_1 and R_2 are combined into R_3, we check whether there is any other rectangle R_4 which is contained in R_3. A rectangle R_4 is contained in R_3 if $u_{s4} \ge u_{s3}$, $u_{e4} \le u_{e3}$, $i_{s4} \ge i_{s3}$ and $i_{e4} \le i_{e3}$. If R_4 is contained in R_3, we remove R_4 from the set of rectangles.

In each iteration of the procedure, referred to as Procedure 2, every pair of rectangles R_{j1}, R_{j2} is considered. If the rectangles can be combined, they are replaced by the combined rectangle R_{j3}. Additional rectangles are then removed if they are contained in R_{j3}. The procedure terminates if no pair of rectangles can be combined in a complete iteration.

3.3. Setting additional 1's in W

In this subsection, we return to the array W defined by Procedure 1, and we describe the setting of additional 1 values in W. These values are selected to ensure that all the detectable circuit faults can be detected on the observed values indicated by W. In the selection process, we use the n-detection information collected by Procedure 1, and a cover of the 0's in W by rectangles obtained using Procedure 2.

Consider the array W at an arbitrary stage of its construction and a fault f which is not detected under W. We say that f is not detected under W if, for every (u,i) such that f is detected at time unit u on output i, $W[u,i] = x$. We observe that the array W already contains some 0 and 1 values selected by Procedure 1, and 1 values selected for faults considered before f. These values require a certain number of rectangles. When adding a 1 value to detect an undetected fault f, we would like to ensure as much as possible that the number of rectangles required to cover the 0 or 1 values in W will not go up. We achieve this as follows. We use Procedure 2 to find a cover of the 0 values in W by rectangles. Let the rectangles found be $R_1,R_2,\cdots,R_m$. If possible, we select for f an entry (u,i) such that f is detected at time unit u on output i, and (u,i) is not contained in any one of the rectangles R_j, $1 \le j \le m$.

For every fault f, the candidate pairs (u,i) are the ones contained in $det(f)$ found by Procedure 1. Of these pairs, we select one which is not contained in any one of the rectangles R_j,

$1 \le j \le m$, and which appears in the maximum number of sets $det(f')$ of undetected faults f'. If no such pair can be found, we select any pair out of $det(f)$ that appears in the maximum number of sets $det(f')$ of undetected faults f'.

The yet-undetected faults are considered in an order such that faults with smaller sets $det(f)$ are considered earlier. For these faults we have fewer options to set values of W to 1 so as to detect the fault. We prefer to consider such faults first in an attempt to minimize the total number of 1's in W.

The procedure, referred to as Procedure 3, uses a simulation process that allows it to determine which faults are detected under a given array W with values $\{0,1,x\}$. This simulation process is applied every time a 1 value is added to W. This is needed since the size of a set $det(f')$ computed by Procedure 1 is limited to n. Thus, if f' is detected more than n times by the *TPG* sequence, it may be detected under W because $W[u,i] = 1$ even if $(u,i) \notin det(f')$.

3.4. Selecting a set of rectangles for *CMP*

Using W obtained at the end of Procedure 3, we apply Procedure 2 to find two sets of rectangles. The set C_0 is a cover of the 0's in W, and the set C_1 is a cover of the 1's in W. The implementation of the *CMP* block is based on the smaller of C_0 and C_1.

4. Experimental results

We applied the synthesis procedure described in Section 3 to ISCAS-89 and ITC-99 benchmark circuits. To investigate the synthesis procedure in the case where all the detectable circuit faults can be detected by the *TPG* sequence, we used test sequences generated by the test generation and test compaction procedures from [8]-[10].

The benchmark circuits we consider do not have tri-state elements that result in unknown output values. In addition, the circuits are synchronized after a small number of time units following which the output sequence is fully specified. To imitate the situation where unknown values exist throughout the output sequence, we injected unknown values randomly into the output sequence of the *CUT*. For this purpose, we considered every time unit u and every output i. With probability p, we changed the value at time unit u on output i from its actual value to a U. We experimented with $p = 0$, 0.005, 0.01 and 0.05.

In the implementation of Procedure 1 we used $n = 20$. The results are shown in Table 1. For every value of p we show the number of 1's in W at the end of Procedure 3, and the number of rectangles required to cover all the 0's or all the 1's of W (whichever is smaller). It can be seen that the number of rectangles is significantly lower than the number of 1's. Thus, we are able to cover large numbers of 1's (or 0's) using each rectangle.

In practice, the percentage of unknown output values in the output sequence of the circuit should be minimized by designing the *TPG* appropriately. Moreover, it is possible to design the *TPG* and the *CMP* block together so as to minimize the size of the *CMP* block. Thus, it is expected that in practice, the *CMP* block will have to deal with relatively small numbers of unknown output values that will fit well into rectangles. As a result, the number of rectangles, which indicates the number of comparison units in the implementation of the *CMP* block, is expected to be relatively low.

In the last column of Table 1 we show the normalized run time of Procedure 3 for the case where $p = 0.05$. Denoting the run time of Procedure 3 by RT_3 and the run time required to fault simulate the test sequence applied to the circuit by RT_0, the normalized run time reported is RT_3/RT_0. The normalized run time provides an indication of the simulation effort in Procedure 3 (the other components of the procedure are relatively fast).

Table 1: Experimental results

	p=0		p=0.005		p=0.01		p=0.05		n.
circuit	1's	rct	1's	rct	1's	rct	1's	rct	time
s298	37	1	37	3	38	3	39	10	13.60
s344	31	1	31	2	31	2	34	8	6.19
s382	52	1	51	7	52	12	50	24	33.58
s400	51	1	52	8	51	13	52	21	35.26
s444	49	1	49	8	49	11	49	24	33.35
s526	45	1	45	9	45	16	44	29	38.52
s641	93	3	94	7	93	9	96	29	24.56
s820	178	1	177	19	182	25	184	66	59.42
s1196	309	1	310	10	310	16	312	53	34.06
s1423	58	1	58	10	61	14	60	35	8.23
s1488	316	1	320	16	324	29	318	84	113.69
s5378	337	5	338	49	343	67	354	115	252.12
b01	11	1	11	1	11	2	11	4	0.56
b02	8	1	8	1	8	1	8	2	0.00
b03	32	1	32	2	32	4	33	11	9.19
b04	93	1	98	4	99	7	99	23	27.17
b06	32	1	32	2	32	3	31	6	5.86
b09	23	1	23	2	23	3	23	5	7.64
b10	40	1	39	4	40	5	40	14	19.82
b11	42	1	44	7	43	9	45	17	16.77

5. Concluding remarks

Complex circuits may produce unknown values in response to an input sequence due to the existence of tri-state elements and due to unknown initial states. Unknown values may appear throughout the output sequence of the circuit. Such unknown values make it difficult to determine the signature of the fault free circuit computed by an output response analyzer which is part of a BIST scheme. We described a procedure for synthesizing a minimal logic block that allows unknown values to be replaced by a known constant while ensuring that the BIST scheme can detect all the detectable faults. The implementation of this logic block was based on building blocks called comparison units. Each comparison unit allows us to avoid a different range of time units where unknown values may appear on a circuit output. The use of comparison units simplifies the synthesis procedure.

References

[1] P. H. Bardell, W. H. McAnney and J. Savir, *Built-In Test for VLSI Pseudorandom Techniques*, Wiley, 1987.

[2] E. B. Eichelberger, E. Lindbloom, J. A. Waicukauski and T. W. Williams, *Structured Logic Testing*, Prentice-Hall, 1991.

[3] V. D. Agrawal, C. R. Kime and K. K. Saluja, "A Tutorial on Built-In Self-Test, Part 1: Principles", IEEE Design & Test of Computers, March 1993, pp. 73-82.

[4] V. D. Agrawal, C. R. Kime and K. K. Saluja, "A Tutorial on Built-In Self-Test, Part 2: Applications", IEEE Design & Test of Computers, June 1993, pp. 69-77.

[5] Y. Zorian, E. J. Marinissen and S. Dey, "Testing Embedded-Core Based System Chips", ITC-98, pp. 130-143.

[6] G. Eide, "Embedded Deterministic Test - DFT Technology for Low-Cost IC Manufacturing Test", www.mentor.com/dft.

[7] I. Pomeranz and S. M. Reddy, "On Synthesis-for-Testability of Combinational Logic Circuits", DAC-95, pp. 126-132.

[8] M. S. Hsiao, E. M. Rudnick, and J. H. Patel, "Sequential Circuit Test Generation Using Dynamic State Traversal", ED&TC-97, pp. 22-28.

[9] R. Guo, S. M. Reddy and I. Pomeranz, "PROPTEST: A Property Based Test Pattern Generator for Sequential Circuits Using Test Compaction", DAC-99, pp. 653-659.

[10] I. Pomeranz and S. M. Reddy, "Vector Restoration Based Static Compaction of Test Sequences for Synchronous Sequential Circuits", ICCD-97, pp. 360-365.