Hybrid Testing Schemes Based On Mutual and Signature Testing

M.F. Abdulla  C.P. Ravikumar  Anshul Kumar
Indian Institute of Technology, New Delhi, 110016, INDIA
E-mail: {abdulla@cse, rkumar@ee, anshul@cse}.iitd.ernet.in

Abstract

Signature based techniques have been well known for the Built-in Self-test of integrated systems. We propose a novel test architecture which uses a judicious combination of mutual testing and signature testing to achieve low test area overhead, low aliasing probability and low test application time. The proposed architecture is powerful for testing highly concurrent systems in applications such as iterative logic arrays, real-time systems, systolic arrays, and low-latency pipelines which tend to have a large number of functional modules of a similar nature. We provide graph-theoretic optimization algorithms to optimize the test area and test application time of the resulting test architecture.

Keywords: Built-in self test, mutual checking, aliasing, self loops

1 Introduction

In this paper, we consider built-in self test (BIST) for data paths with concurrency, e.g. real-time systems, systolic arrays, and low-latency pipelines. BILBO-BIST is a popular method for testing these data paths but is is associated with some inherent problems such as Aliasing. When BILBO-BIST is applied to concurrent data paths, the test application time is large since a large number of test sessions will be required to test all the modules. Although the initial few test patterns may detect a fault in the module under test, it is not until the end of a large number of test patterns that the possibly faulty signature is scanned out and compared. Such a problem can waste a large number of test cycles. The long signature scan-out time is another problem associated with BILBO-BIST.

A technique to reduce aliasing is multiple signature checking [1]. Approaches to eliminate aliasing by avoiding output compaction by implementing duplicate modules and comparators in the chip where the output test responses of a module and its duplicate are compared together [5]. Such techniques are usually applied in on-line testing and the area overhead for such architectures is high.

In this paper, we propose MC-BIST (Mutual Checking-BIST), a new BIST architecture which uses Mutual Checking without duplicating the modules. It also uses Multiple Signature Checking to a limited extent. The proposed scheme reduces the probability of aliasing and ensures that any faulty response can be observed as soon as it appears. The total test application time is reduced by improving the test concurrency. The area overhead of the MC-BIST architecture is minimum and mainly introduced by the implementation of a number of auxiliary interconnects which improve the test concurrency. The paper has been organized as follows. In Section 2, we explain the concept of mutual checking scheme. We also describe techniques to further improve the fault masking probability of the proposed scheme. In Section 3, we describe Area oriented and Test-time oriented graph-theoretic optimization algorithms. Results on benchmark circuits are given. Conclusions are presented in Section 4.

2 Mutual Checking Scheme

Let us consider two functionally identical modules \( M_1 \) and \( M_2 \). Let \( X = x_0 x_1 \ldots x_n \) and \( Y = y_0 y_1 \ldots y_n \) be the output responses of \( M_1 \) and \( M_2 \), respectively for the same set of \( n \) test vectors. If the responses \( x_i \) and \( y_i \) are compared, then the output of the comparator, called error signal, will remain low if \( x_i = y_i \) for \( 1 \leq i \leq n \). If \( x_i \) and \( y_i \) are different at the time instance \( t_i \), then the error signal becomes high at time \( t_i \). In this event, the testing process is terminated and the chip is declared faulty. Figure 1 illustrates the concept of mutual checking which forms the basis of the proposed test architecture. A module for which there does not exist another functionally equivalent module is tested using signature comparison scheme [1, 21.

2.1 The Basic Idea

At the outset, it appears that the generation of each error signal requires a separate equality comparator. However, we show that a simple modification to the BILBO register can be used to generate the error signal. We call this modified register as an Equality Com-

![Figure 1: Mutual checking.](image-url)
paritor BILBO (E-BILBO). Here, each stage of the BILBO register have an additional 2-input AND, and 2-input OR gates. Some other gates are also inserted to control the internal state of the E-BILBO register. See Figures 2 and 3.

Figure 2: E-BILBO block diagram.

There are 4 modes of operation in the E-BILBO register, as controlled by the signals A and B. During normal operation, an E-BILBO serves as a parallel-in parallel-out register (AB=10); in the test mode, an E-BILBO can be configured into a pseudo-random pattern generator (AB=01), into a comparator AB=11, or into a serial-in serial-out shift register 1A2B=00. In a testable data path, the E-BILBOs are connected to the scan path along with other test registers for the purpose of initialization. The area overhead required to transform an m-bit BILBO into an m-bit E-BILBO in static CMOS technology is less than 12% of the BILBO area.

2.2 E-BILBO as a comparator

In comparing two sequences X and Y of test output vectors in an E-BILBO, we should guarantee that there is a delay of one clock cycle between them. Figure 4(a) shows the configuration of the E-BILBO register in the comparator mode. Figure 4(b) shows the sequence of comparisons of X and Y. Initially, yo is the initial seed of the E-BILBO. At time 0, a0 and y1 are input to the E-BILBO. a0 is compared with yo and the result of the comparison is observed by the error signal of the E-BILBO. At the same time, y1 is stored in the E-BILBO. In the next time step, 0 is compared with the content of the register and y2 stored. The MC-BIST scheme between two identical modules a1 and a2 is applicable only if both the modules are of the same speed. In case where one of the functionally identical module is faster than the other, the test clock for this module is slowed down. An auxiliary interconnect is required to make the output response from a2 available to the E-BILBO register. This interconnect does not serve any purpose during the normal mode of operation of the chip and is only useful during testing. Thus the additional interconnect only affects the area requirement of the chip and not the delay performance of the circuit during normal operation. An optimization which is possible is to share a single interconnect bus for realizing two or more auxiliary interconnects, possibly at the cost of reduced concurrency in testing.

2.3 Fault Masking

It is clear that when two identical modules are tested through the mutual checking scheme, there will be no fault masking if only one of these modules is faulty, or both are faulty with non-equivalent faults. Since the physical design of two functionally identical modules is performed by instantiating the same module type twice, an error in the module design may show up as a fault in both the instances [2]. Such a duplicated fault is called a common design fault. In the following, we describe three techniques to reduce the probability of fault masking in the presence of a common design fault.

Avoiding Common Design Faults : The adverse effect of a common design fault on the effectiveness of the MC-BIST scheme can be avoided by providing different structural realizations for the two functionally identical modules. In practice, component libraries have multiple implementations of the same component.

Multiple Mutual Checking : The probability of a design fault arising in all the k replications of the same module becomes smaller as k increases. Therefore, testing more than two identical modules together through mutual checking scheme can greatly reduce the probability of masking. We refer to this technique as multiple mutual checking (MMC).

Mutual Checking with Compaction : In this scheme, a pair of identical modules are tested using the MC-BIST scheme; in addition, one of the modules is
tested using signature testing. We refer to this scheme as Mutual Checking with Compaction (MCC).

3 Optimization of Test Architecture

3.1 Merging Embeddings

A structure consisting of the module \( mi \) and its associated PRPGs, MISR and the intermediate logic is called a BILBO Test Embedding \( bte_1 = (R, L', \{r, L''\}) \). \( R' \) is the set of registers configured as PRPG, \( L' \) is the intermediate logic between \( R' \) and \( m_i \), \( r \) is the BILBO register configured as MISR, and \( L'' \) is the intermediate logic between \( m_i \) and \( T. \) Let \( BTE = bte_1, bte_2, \ldots, bte_x \) be the set of the test embeddings of the circuit functional modules. We define a mutual test embedding as a 3-tuple which includes (1) a set of test patterns, (2) two functionally identical modules, and (3) an E-BILBO register configured in the comparator mode.

Definition 1 Let \( bte_1 = (R_1, L_1, m_1, T_1, L_1') \) and \( bte_2 = (R_2, L_2, m_2, T_2, L_2') \) be two BILBO test embeddings. A mutual test embedding \( mte_{12} = bte_1 \# bte_2 \) is defined \( BS \{ \{R_1, L_1, m_1, T_1, L_1'\}, \{R_2, L_2, m_2, T_2, L_2'\} \} \). Com is the register \( r_{12} \) which is configured as E-BILBO in comparator mode.

The two test embeddings \( bte_1 \) and \( bte_2 \) can be merged into a mutual test embedding if and only if the following four conditions are satisfied. (1) The modules \( m_1 \) and \( m_2 \) are functionally identical. (2) There is no common intermediate logic between the PRPGs and the modules under test, i.e., \( L_1' \cap L_2' = \emptyset \). (3) The signature register \( T \) is not used as an MISR in a test embedding of a module \( m_i \), for which there is no functionally identical module, and (4) The signature register \( r \) is not in \( R_1 \).

To implement the mutual checking with compaction, in addition to the above 4 conditions the output port of the module \( m_i \) should be connected to a BILBO register which will be configured as an MISR. Let \( MTEi = \{mte_1, mte_2, \ldots, mte_x\} \) be the set of \( j \) mutual test embeddings generated for a module type \( i \). \( MTE_i \) may be empty. Let \( MTE = MTE_1, MTE_2, \ldots, MTE_x \) be the set of subsets of mutual test embeddings, where \( t \) is the number of the modules types in the circuit. In the set \( MTEi \), the conditions to merge two mutual test embeddings \( mte_{12} = bte_1 \# bte_2 = (L_1', L_2', \{r_{12}, L_{12}'\}) \) and \( mte_{12} = bte_1 \# bte_2 = (L_1', L_2', \{r_{12}, L_{12}'\}) \) into MCC scheme are: (1) If \( M_i M_j \neq 8 \), or (2) If \( M_i M_j \neq 8 \), and the merge \( bte_1 \# bte_2 \) is permissible.

There are two ways to implement the MC-BIST scheme (a) Area oriented optimization where priority is given to mutual checking scheme for all modules compatible with this scheme with the intention of minimizing the test area overhead. (b) Test-time oriented optimization which is an optimized implementation to keep the test area overhead within limits while reducing the total test time.

3.2 Area Oriented Optimization

For each module type \( i \), we construct a Merge-compatibility graph \( MCG(S, E; i) \) as follows. The nodes in \( V \) correspond to the BILBO test embeddings for the modules of type \( i \). An edge \((e_1, e_2)\) is included in \( MCG \) if the embeddings \( e_1 \) and \( e_2 \) can be merged, i.e., \( e_1 \# e_2 \) is permissible. A weight \( w_{i_1}l_2 \) is associated with edge \((e_1, e_2)\) which is the weight of aux line associated with the mutual test embedding \( e_1 \# e_2 \).

Consider a connected subgraph \( G \) in \( MCG \). Let \( T \) be the minimum spanning tree in \( G \) and let \( Cost(T) \) be the sum of the edges weight in \( T \). The modules corresponding to the nodes in \( G \) can be tested in the "multiple mutual checking" mode. The problem of deriving a test architecture for the chip can now be stated as follows: Step 1, Identify a merge compatibility graph for each module type. Step 2, In the \( MCG \), find \( k \) minimum spanning trees \( T_1, T_2, \ldots, T_k \) which span subgraphs \( GI, G_2, \ldots, G_k \) of \( MCG \). Ensure that \( GI \cap G_m = \emptyset \). Step 3, cost \( Z \) is minimum.

The modules in \( GI \) are tested using the multiple mutual checking mode if there are more than \( n \) nodes in \( GI \). When there are exactly \( n \) nodes in \( GI \), the simple mutual checking architecture is applicable. When there is only one node in \( GI \), we use the BILBO test embedding for the module. The above graph optimization problem is hard since the partition of \( MCG \) into \( GI, G_2, \ldots, G_k \) is not known a priori. Special cases of the optimization problem permit polynomial-time solution. For example, if \( MCG \) is a connected graph and \( k = 1 \), the optimization problem boils down to finding the minimum spanning tree in \( MCG \). If the number of nodes in \( MCG \) is even, say \( 2C \), and the number of subgraphs \( k = C \), then the optimization problem simplifies to finding a minimum weighted matching in \( MCG \). In general, the "Tree Cover Problem" defined above is a hard problem and we have develop a heuristic algorithm to solve the problem. This algorithm finds an approximate solution to the tree cover problem by first computing a minimum weighted matching \( E \) in \( MCG \). If a node \( U \) in \( MCG \); is left uncovered by the matching, then the algorithm selects an edge \((U, z)\) of minimum weight and includes it in \( E \).

Multiple Mutual Checking: A k-way MMC corresponds to \( k \) modules being tested together in the multiple mutual checking mode. We can therefore redefine the tree cover problem so as to generate MMs of \( k \) or larger. However, since large values of \( k \) imply large test area overhead, we try to restrict \( k \) to be in the range 3, 4, or 5. The k-ways MMC with \( k > 6 \) can be decomposed into smaller MMCs by eliminating selected edges from the tree.

3.3 Test Scheduling

Let \( MTE \) be the set of selected mutual test embeddings. The modules which are not covered by \( MTE \) must be tested using BILBO test embeddings. Well known algorithms from the literature [4] can be used to select a cost-effective set of BILBO test embeddings; let \( BTE \) denote such a selected set. Let \( QTE = MTE \cup BTE \). Our next objective is to come up with a test schedule which is as concurrent as possible so that the total test time is minimized. We create an embedding compatibility graph \( ECG \), where the modules correspond to embeddings from \( QTE \). An edge \((e_1, e_2)\) in the \( ECG \) implies that embeddings \( e_1 \) and \( e_2 \) are concurrency compatible. Many partitioning algorithms for the compatibility graphs have been reported in [3, 4].
3.4 Test-Time Oriented Optimization

In this section, we consider how to make use of a combination of BILBO testing and mutual testing in order to optimize the total test application time. Minimization of test area overhead is a secondary objective in this process.

There are two transformations which can reduce the test application time for a given test plan. (1) Embedding Execution Shifting: Let \( M \) be the module which has the largest number of test vectors applied during session \( i \). Let \( \text{bteM} \) be the BILBO test embedding for module \( M \). The transformation is to reschedule the test embedding \( \text{bteM} \) in a session \( j \neq i \), by merging \( \text{bteM} \) with an embedding from session \( j \). (2) Session Elimination: A test session \( i \) can be eliminated if all BILBO test embeddings scheduled in session \( i \) can be merged with test embeddings from other sessions using embedding execution shifting. We can generalise the above ideas through the help of the following theorem.

**Theorem 1** Given a set \( V \) of BILBO test embeddings. Let \( G(V,E) \) and \( G(V,E_o) \) be two CO71CUT-Covency compatibility graphs. Further let \( E_1 \cap E_o \). Let \( TT\text{ency} \) and \( T_2 \) be the optimum test application time for graphs \( d \) and \( G_o \). Then \( T_2 \leq T_1 \).

The above theorem emphasizes the need to enhance a given concurrency compatibility graph with additional given edges which enhance the mutual test embeddings. We first construct a test concurrency compatibility graph from the BILBO test embeddings. The compatibility relations is then enhanced by adding more edges between incompatible nodes to reflect the possibility of a selective incompatibility merging. We generalise the above ideas through the help of the following theorem.

**Cost**

\[ \text{Cost}_{b(i)} + \text{Cost}_{t(i)} + \cdots + \text{Cost}_{s(i)} \leq \text{Bound} \]

Using known algorithms [3, 4], we compute a minimum clique cover of the concurrency compatibility graph. Such a clique cover is the solution to the time-oriented optimization problem.

Table 1 gives the percentage of reduction in test time \( T_{id} \) the degree of the implementation of the mutual checking for different data paths. \( T_{id} \) is the saving in the test time compared to the SSCS scheme when all the chip modules have the same testing time and \( T_{add} \) is when the testing times of the multiplier and the adder/subtractor modules are assumed to be 5 units and 2 units, respectively. Column 5 gives the area overhead of the implementation of the mutual checking. Columns 6 gives the percentage of the area overhead of the composite SSCS and MC-BIST scheme compared to the area overhead of the basic SSCS scheme. Columns 7 gives the percentage of the area overhead of the composite MOSC and MC-BIST scheme compared to the area overhead of the basic MOSC scheme.

<table>
<thead>
<tr>
<th>Data Path</th>
<th>FB1</th>
<th>% Time Save</th>
<th>MC-B</th>
<th>% MC-BIST Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dih</td>
<td>1</td>
<td>33</td>
<td>17</td>
<td>18</td>
</tr>
<tr>
<td>Dih</td>
<td>3</td>
<td>0.0</td>
<td>25</td>
<td>15</td>
</tr>
<tr>
<td>Dih</td>
<td>3</td>
<td>0.0</td>
<td>0.0</td>
<td>1.8</td>
</tr>
<tr>
<td>Dih</td>
<td>4</td>
<td>33</td>
<td>42</td>
<td>73</td>
</tr>
<tr>
<td>Diff</td>
<td>5</td>
<td>33</td>
<td>42</td>
<td>91</td>
</tr>
<tr>
<td>CRT#1</td>
<td>1</td>
<td>33</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>CRT#2</td>
<td>2</td>
<td>0.0</td>
<td>33</td>
<td>67</td>
</tr>
<tr>
<td>CRT#3</td>
<td>3</td>
<td>0.0</td>
<td>20</td>
<td>100</td>
</tr>
<tr>
<td>CRT#4</td>
<td>3</td>
<td>33</td>
<td>53</td>
<td>83</td>
</tr>
<tr>
<td>Hall1</td>
<td>1</td>
<td>25</td>
<td>14</td>
<td>22</td>
</tr>
<tr>
<td>Hall2</td>
<td>2</td>
<td>50</td>
<td>50</td>
<td>44</td>
</tr>
<tr>
<td>Hall3</td>
<td>4</td>
<td>50</td>
<td>29</td>
<td>89</td>
</tr>
<tr>
<td>Hall4</td>
<td>5</td>
<td>50</td>
<td>36</td>
<td>100</td>
</tr>
</tbody>
</table>

4 Conclusions

In this paper, we propose a novel Mutual Checking BIST (MC-BIST) architecture. The proposed architecture is powerful for testing highly concurrent data paths in applications such as iterative logic arrays, systolic arrays, real-time systems and low-latency pipelines. Data paths can be made testable using a BIST structure called Equality comparator Register (E-BILBO) proposed in this paper. Apart from reducing the aliasing probability, the MC-BIST test architecture also reduces the total testing time. A number of techniques were proposed to further reduce the aliasing probability through a combination of mutual testing and signature testing. The proposed architecture resulted in up to 50% savings in total test time overhead for some High-level synthesis benchmark circuits with very little area overhead.

References