

A Systematic Approach for Designing Testable VLSI Circuits

Sen-Pin Lin, Charles A. Njinda and Melvin A. Breuer
Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, CA 90089-0781

Abstract

A systematic approach has been developed to provide designers with a set of testable versions for a given design, ranging from the minimal test time solution to the minimal area overhead solution. The designer thus has the flexibility to make the necessary trade-off between area overhead and test time depending on the constraints of the design under consideration. By employing an expert selection system developed previously, the system can be extended to operate as an intelligent BIST design advisor. Experiments have been performed on several circuits generated by MABAL, a high-level synthesis tool, to demonstrate the performance of this approach.

Index terms: Built-in-self-test, synthesis for testability, test scheduling.

1 Introduction

Numerous design for testability (DFT) and built-in self-test (BIST) techniques are used to make circuits testable. The Built-In Logic-Block Observation (BILBO)[2] BIST technique requires modifications to be made to a circuit so that each combinational logic block (*kernel*) is fed directly or indirectly by a pseudo random pattern generator (PRPG) and directly or indirectly feeds a signature analyzer (SA). This extra hardware increases the layout area. In the test mode, a sufficient number of test patterns need be generated to achieve acceptable fault coverage. Therefore, *area* and *test time* are two important overheads encountered when using the BILBO technique. Reducing these overheads without sacrificing fault coverage is important. Unfortunately, most existing systems[1, 5, 6] make a design testable by trying to minimize either test time or area overhead. The solutions generated are often not globally optimal. A designer should have the flexibility to make trade-offs between area overhead and test time depending on the design constraints. The procedures presented in this paper allow such a trade-off to be made.

Abadir and Breuer[1] proposed a Testable Design Expert System (TDES) which allows a user to select an embedding for each kernel based on the design constraints. However, TDES only considers a local view

*This work was supported by the Defense Advanced Research Projects Agency and monitored by the Federal Bureau of Investigation under Contract No. JFBI90092.

of each kernel, thus its performance depends on the order in which the kernels are processed. We too allow user interactions for making choices, but a global view in selecting embeddings is taken to achieve better performance. Craig *et al.*[6] formulated a clique covering problem to minimize test time. The restriction that resource allocation is done before test scheduling makes the optimality of the solution applicable only to a specific resource allocation and thus the solution may not necessarily be a global optimum. Bhawmik *et al.*[5] used an integer linear programming technique to minimize area overhead of a BIST design. Our study of integer linear programming shows little success for this problem, and a branch and bound technique is suggested instead.

2 Preliminaries

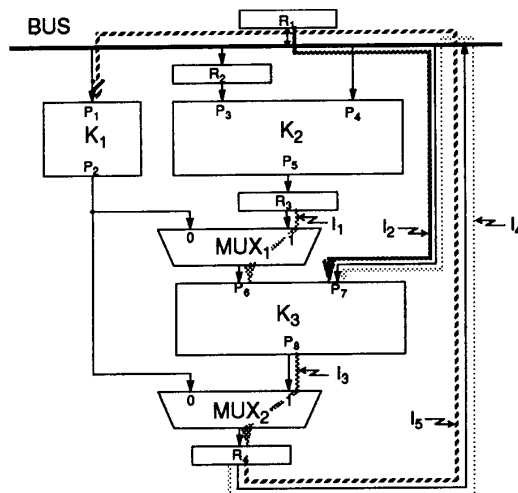


Figure 1: An example circuit

Consider a circuit consists of kernels, registers, buses and MUXes. A *K-port* is an input/output port of a kernel. An *I-path*[1] is a data path from a primary input or a register to a *K-port* or from a *K-port* to a primary output or a register so that data can be transferred unaltered. In this paper, an *I-path* is represented as a list (c_1, c_2, \dots, c_n) where each c_i , $1 \leq i \leq n$, is a component or a port of a component in the circuit, and data can be transferred from c_{i-1} to c_i without changes.

Some I-paths in the example circuit of Figure 1 are $I_1 : (R_3, MUX_1(1), K_3(P_6))$, $I_2 : (R_1, BUS, K_3(P_7))$, $I_3 : (K_3(P_8), MUX_2(1), R_4)$, $I_4 : (R_4, BUS, K_3(P_7))$ and $I_5 : (R_4, BUS, K_1(P_1))$.

Definition 1 A register is a *driver* of a K-port P_i if there exists an I-path from it to P_i ; a register is a *receiver* of a K-port P_j if there exists an I-path from P_j to it.

A *test session* is defined as the execution of a set of tests for some kernels. Only one test session can execute at a time. For each test session a *test plan* is required. This plan specifies how to initialize test hardware, perform tests for kernels, and observe final signatures. It is possible that a register can be a driver of some K-port and a receiver of another K-port. In the BILBO methodology a register is not allowed to be used as both a driver and a receiver in the same test session.

Definition 2 An I-path is a *driving path* if it starts from a primary input or a driver and ends at an input K-port; an I-path is a *receiving path* if it starts from an output K-port and ends at a receiver or a primary output.

By these definitions, I_1 , I_2 , I_4 and I_5 are driving paths; I_3 is a receiving path. Driving or receiving paths are said to *cover* their associated K-ports. A K-port is said to be *testable* if it is covered by at least one I-path. To make a K-port testable using some I-path that covers it, it may be necessary to add extra functions to some of the registers in the I-path like pseudo random pattern generation (PRPG), signature analysis(SA), HOLD, or LOAD if these functions are absent. If there is no I-path that can cover a K-port, extra registers or MUXes may be needed. To shift in the initial seed data and shift out the final signature, the SHIFT function needs to be added to all registers that operate as PRPGs or SAs. By adding functionality to a register, the area required for its implementation increases.

Definition 3 Two I-paths are said to have a *Forbidden Conflict (FC)* if they cannot co-exist in a testable design; two I-paths are said to have a *Hard Conflict (HC)* if they cannot be operated in the same test session; two I-paths are said to have a *Soft Conflict (SC)* if they can be operated in the same test session but restrictions on their schedules exist.

For the circuit in Figure 1, I_3 and I_4 have a FC since R_4 is being used simultaneously as a PRPG and a SA to test K_3 . I_3 and I_5 have a HC since R_4 must be used in different test sessions; once as a SA for K_3 , then as a PRPG for K_1 . I_2 and I_5 have a SC due to the sharing of BUS. A set of rules[8] have been formulated to identify all possible configurations that may cause I-path conflicts. A kernel is said to be *testable* if each of its K-ports is testable and no conflicts exist between the driving and/or receiving paths of its K-ports. A circuit is said to be *testable* if each of its kernels is testable.

3 Exploring the solution space

There are usually many ways to make a non-trivial circuit testable. Each testable design has a certain *area overhead* and *test time*. The design with minimal test time is denoted by s_{mtt} ; the design with minimal area overhead is denoted by s_{mao} . Between these two de-

signs can exist a set of intermediate testable designs S_{int} . This family of solutions represent the choices for making a circuit testable.

3.1 The minimal test time solution

Definition 4 A *legitimate BILBO embedding (embedding for short)* is a structure consisting of a kernel, a driving path for each input K-port and a receiving path for each output K-port of the kernel. No FCs or HCs can exist between pairs of these paths.

In the circuit shown in Figure 1, $e = (K_3, I_1, I_2, I_3)$ is a legitimate embedding. K_3 is the kernel of e , I_1 and I_2 are the driving paths for P_6 and P_7 , respectively, and I_3 is the receiving path for P_8 .

A kernel may be contained in several embeddings. These embeddings are obtained by combining the I-paths that have no HCs or FCs. To achieve minimal test time, *pipeline* structures and *minimal latency*[4] can be used, where minimal latency is the minimal delay between two consecutive initiations of test vectors so that no resource conflicts can occur. The minimal latency can always be achieved by using NO-OPs [1][4] in the test plan. If the I-paths of an embedding do not have SCs, the minimal latency is always 1, i.e., a new test vector can be applied every clock period. If a kernel needs T test vectors, then the test time is approximately T clock periods if T is large, i.e. we ignore the time to shift in(out) the initial(final) states of the various shift registers.

We assume that each kernel requires T test vectors. Two embeddings are *fully compatible* if the I-paths between them do not have conflicts. They can be executed in the same test session to reduce the total test time. When the I-paths between two different embeddings have SCs, reservation tables[4] for their test plans have to be analyzed to obtain the test time for executing them in one test session, and the test time for executing them in sequence[8]. If the former is shorter, these two embeddings are said to be *partially compatible*. Only fully and partially compatible embeddings will be considered for concurrent execution.

3.1.1 An optimal procedure

Definition 5 An embedding or a set of pair-wise fully compatible embeddings is called a *Fully Compatible Embedding set(FCE)*. A set of embeddings is called a *Partially Compatible Embedding set(PCE)* if every pair of the embeddings is either fully compatible or partially compatible, but they are not all pair-wise fully compatible.

Clearly, the test time for executing embeddings in a FCE/PCE in one test session is shorter than that of executing them in sequence.

Definition 6 A set of embeddings is called a *Maximal Compatible Embedding set(MCE)* if it is a FCE or PCE and is not a proper subset of any other MCEs.

MCEs can be obtained by a procedure modified from a classical algorithm for finding maximal compatible sets of states in the state minimization problem[3]. A kernel is said to be *covered* by a FCE/PCE if it is the kernel of an embedding which belongs to the FCE/PCE. A kernel can be covered by more than one

FCE/PCE. A solution of the testable design problem is represented by a set of *FCEs* and *PCEs* so that every kernel is covered at least once. Each *FCE/PCE* in the solution needs one test session and s_{mtt} is a solution whose cumulative test time of the *FCEs* and *PCEs* is minimal. A set of *MCEs*, denoted by \mathcal{M} , is said to be *reducible* to a solution if every *FCE/PCE* in this solution is a subset of some *MCE* belonging to \mathcal{M} .

A conventional covering algorithm can be adopted to find a minimal *MCE* cover[6], but this *MCE* cover is not necessarily optimal if some kernels are covered more than once. An efficient algorithm using the branch and bound technique has been developed to first search for a reducible *MCE* set and then reduce it to a minimal test time solution. Pruning techniques are used to reduce the search space.

A reducible *MCE* set can usually be reduced to different solutions. For example, suppose there are three kernels K_1, K_2 and K_3 in a circuit. Assume that MCE_1 covers K_1 and K_2 and MCE_2 covers K_1 and K_3 . The *MCE* set $\{MCE_1, MCE_2\}$ can be reduced to a solution where K_1 and K_2 are tested in one test session and K_3 is tested alone in the second session. However, it can also be reduced to the solution where K_2 is tested in one test session and K_1 and K_3 are tested in the other session. From a reducible *MCE* set, only the solution reduced from it with the shortest test time will be considered.

A sketch of procedure *min.time1*, which generates a minimal test time solution, is presented below. More details can be found in[8]. When two solutions have the same test time, the design with less area overhead is selected.

Procedure *min.time1*

Input: All kernels and *MCEs* in a circuit.

Output: A minimal test time solution.

1. Let \mathcal{K} be the set of all kernels.
2. Select a kernel from \mathcal{K} to be covered.
Select a *MCE* which covers this kernel.
Remove kernels covered by the *MCE* from \mathcal{K} .
3. If the current test time exceeds the best test time so far, then backtrack.
4. If \mathcal{K} is not empty, then goto 2.
5. Reduce the *MCE* set to a solution. If the test time is shorter than the best test time so far, mark the solution as the best solution, update the best test time and backtrack.

3.1.2 A sub-optimal procedure

The computation time for procedure *min.time1* can be excessive for moderate sized circuits due to the extremely large number of *MCE* sets. To reduce computation time, a greedy procedure *min.time2*[8] can be used. In this procedure, embeddings instead of *MCE* sets are used to cover kernels in the search process. We implicitly enumerate the combinations of embeddings so that every kernel is covered once. For every such combination, a greedy approach is used to schedule the embeddings so as to minimize test time.

A partial schedule *PS* consists of a set of test sessions, where each test session contains a set of pair-wise fully/partially compatible embeddings. To schedule an embedding e , we will find the first test session in *PS*

so that e is compatible with every embedding in this test session. If such a test session exists, e is added to it. If e cannot fit into any of the test sessions in *PS*, a new test session is created which contains only e .

The greedy approach described cannot guarantee finding the shortest test time solution. However, in the experiments we have carried out to date it has always generated an optimal solution.

3.2 Minimal area overhead solution

With the information on I-path conflicts, an efficient procedure *min.area* based on the concept of branch and bound is developed to generate a minimal area overhead solution. A set of I-paths that are not allowed to be used due to I-path conflicts is kept in \mathcal{I} during the process. \mathcal{I} becomes larger as more kernels are covered. A sketch of procedure *min.area* is given next; more details and some examples can be found in [8].

Procedure *min.area*

Input: All kernels and I-paths in a circuit.

Output: A minimal area overhead solution.

1. $\mathcal{I} = \emptyset$.
2. Select a K-port to be covered.
Select an I-path, I_k , which covers this K-port.
Add the I-paths which have FCs with I_k to \mathcal{I} .
If the current area overhead exceeds the best area overhead so far, then backtrack.
3. If every K-port has been covered, then mark this solution as the best solution, update the best area overhead and backtrack, else goto 2.

3.3 Intermediate solutions

In general the area overhead of s_{mtt} may be too high and the test time of s_{mao} may be too long. Neither is acceptable if both attributes are important. To develop an intelligent and user-friendly testable design system, it is important to provide a designer with a family of solutions having different test time and area overhead.

Notice that if no pruning is done, either procedure *min.time1* or *min.time2* is capable of generating almost all possible solutions. To make the problem more tractable, solutions can be partitioned into groups, where solutions with similar test time are put into the same group. In a group of solutions only the one with smallest area overhead is considered, which is called a *representative solution*. Let $t1$ be the test time of s_{mtt} and $t2$ be the test time of s_{mao} . Define $\Delta t = \frac{t2-t1}{G}$, where G is the number of representative solutions desired. If \mathcal{N} is the set of solutions with test time no greater than $t2$, then we can partition \mathcal{N} into G groups, where group i contains solutions with test time in the interval $(t1 + (i - 1) \times \Delta t, t1 + i \times \Delta t]$.

A procedure *min.int*[8], which is slightly different from procedure *min.time2*, is developed to generate a set of representative solutions. Instead of keeping only one solution, we will retain a best solution for every group. A relaxed bounding technique is used so that only solutions with test time exceeding $t2$ are pruned.

4 Experimental results

For the circuit shown in Figure 1, the minimal test time solution and the minimal area overhead solution are summarized below.

Minimal test time solution
Test time: 2T Area overhead: 9

Test session #1	Test session #2	
K_3	K_1	K_2
R_3 drives P_6	R_1 drives P_1	R_2 drives P_3
R_1 drives P_7	R_4 receives P_2	R_1 drives P_4
R_4 receives P_8		R_3 receives P_5

Minimal area overhead solution
Test time: 3T Area overhead: 7

Test session #1	Test session #2	
K_3	K_1	K_2
R_3 drives P_6	R_1 drives P_1	R_1 drives P_3
R_1 drives P_7	R_2 receives P_2	R_1 drives P_4
R_2 receives P_8		R_3 receives P_5

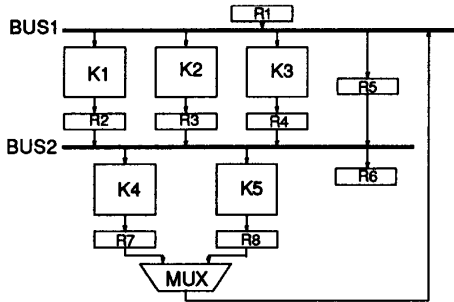


Figure 2: Example circuit for Table 3

We assume that initially every register has only the LOAD function and the area overhead of adding either a PRPG or SA function to any register is 2 units, while adding both costs 3 units. Due to the simplicity of the circuit, no intermediate solutions are found. We have also carried out several experiments using circuits generated by MABAL (a Module And Bus ALlocator)[7] (See Table 1). The results are summarized in Table 2.

Circuit	#kernel	#MUX	#reg.	#I-path	#emb.
DAC4	10	19	11	43	21
DAC8	6	18	7	39	33
DAC10	4	12	9	30	36
DAC16	3	9	8	30	92

Table 1: Summary of circuits

Circuit	s_{mtt}		s_{mao}		S_{int}		CPU time (sec)
	TT	AO	TT	AO	TT	AO	
DAC4	3T	28	5T	26	4T	27	3.4
DAC8	2T	14	3T	12	NA	NA	5.7
DAC10	1T	18	3T	13	2T	14	4.7
DAC16	1T	10	3T	6	2T	8	3.6

TT - Test time, AO - Area Overhead, NA - not applicable

Table 2: Summary of the results

The CPU time consists of both pre-processing time (i.e. finding I-paths, embeddings, etc) and the run time of the procedures as measured on a Sun-4/490. The assumption that every kernel requires the same number of test vectors greatly reduces the number of representative solutions. For the purpose of illustration, we also process a circuit shown in Figure 2. The results are summarized in Table 3.

solution	test time	area overhead
s_{mtt}	1T	12
intermediate solution #1	3T	11
intermediate solution #2	4T	8
intermediate solution #3	6T	6
s_{mao}	7T	4

Table 3: Solutions of circuit in Figure 2

5 Conclusion and future research

We have presented a systematic approach to generate a set of testable versions of a design using the BILBO test methodology. The space of testable designs is explored so that the minimal area overhead, minimal test time, and intermediate solutions are provided to a designer for consideration. Instead of separating the resource allocation and test scheduling problems as is done in most synthesis systems, our approach deals with them at the same time. Hence a solution contains all the information about extra hardware required, area overhead, test schedule, and test time. No pre-processor[6] or post-processor[5] are needed.

Though the BILBO architecture is the only TDM considered in this paper, extension to other TDMs can easily be dealt with. Trade-off between area overhead and test time is the unique feature of this system. Other design considerations such as extra I/O pins and fault coverage will be considered in the near future. In the future we will consider the case where the number of test vectors for each kernel may not be all the same.

References

- [1] M.S. Abadir and M.A. Breuer. A Knowledge-Based System for Designing Testable VLSI Chips. *IEEE Design & Test*. Aug. 1985, pp. 56-68.
- [2] B. Konemann, J. Mucha, and G. Zwihoff. Built-In Logic Block Observation Technique. *Digest of Papers 1979 Test Conf.* Oct. 1979, pp. 37-41.
- [3] F.J. Hill and G.R. Peterson. *Introduction to Switching Theory and Logical Design*. Wiley, New York, 1981.
- [4] K.Hwang and F.A.Briggs. *Computer Architecture and Parallel Processing*. McGraw-Hill, New York, 1984.
- [5] P.R. Chalasani, S. Bhawmik, A. Acharya and P. Palchadhuri. Design of Testable VLSI Circuits with Minimum Area Overhead. *IEEE Trans. on Computer*. Vol.C-38, Oct. 1989, pp.1460-1462.
- [6] G. Craig, C. Kime, and K. Saluja. Test Scheduling and Control for VLSI Built-In Self-Test. *IEEE Trans. on Computers*. Vol. C-37, Sept. 1988, pp. 1099-1109.
- [7] K. Kucukcakar and A.C. Parker. MABAL: A Software Package for Module and Bus Allocation, *Int'l J. of Computer Aided VLSI Design*. Vol.2, 1990, pp. 419-426.
- [8] S.P. Lin, C.A. Njinda and M.A. Breuer. *A Systematic Approach for Designing Testable VLSI Circuits*. Tech. Rpt. CRI-91-18, Computer Research Inst. (EE Dept.) Univ. of Southern Calif., Los Angeles, July 1991.