Test Session Oriented Built-in Self-testable Data Path Synthesis

Han Bin Kim[1], Takeshi Takahashi[2], and Dong Sam Ha[1]

[1]Bradley Dept. of Electrical and Computer Engineering
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061-0111

[2]Advantest America R&D Center Inc.
3201 Scott Blvd
Santa Clara, CA 95054

Abstract

Existing high-level BIST synthesis methods focus on one objective, minimizing either area overhead or test time. Hence, those methods do not render exploration of large design space, which may result in a local optimum. In this paper, we present a method which aims to address the problem. Our method tries to find an optimal register assignment for each k-test session. Therefore, it offers a range of designs to the designer with different figures of merit in area and test time. Experimental results show that our method performs better than or comparable to existing BIST synthesis systems.

I. INTRODUCTION

The design process of a digital system usually starts at a behavior level and descends to a structural level. Test synthesis is to incorporate design-for-testability features in the design process. Until the end of 80's, testability was usually inserted into a structural level (mostly at the gate level) as a post processing of the logic design. However, the approach often fails to yield a good solution (in terms of hardware overhead, fault coverage and testing time) due to an inappropriate choice of hardware structure made in an earlier design stage.

In order to address the problem, researchers investigated the incorporation of testability into the front-end of the design process called high-level test synthesis [1]-[27]. High-level synthesis is to transform a behavioral description of a design into a structural implementation comprised of data path logic and control logic [13]. High-level test synthesis incorporates some testability feature(s) during the high-level synthesis, and the resultant circuit is easier to test than the original circuit in which testability is not considered. An excellent survey for high-level test synthesis is available in [14].

Depending on the testability schemes incorporated, high-level test synthesis systems can be classified into three groups. The first group of high-level test synthesis systems aims to improve the controllability and/or the observability of the circuit [1], [3], [6], [15], [20]. The methods are ones such as insertion of test points, minimization of cycles and/or sequential depth. The second group of high-level test synthesis systems incorporates the scan technique [16], [17], [19]-[23]. While incorporating the scan technique, the systems try to minimize sequential depth and/or cycles, and maximize the number of input and output registers. The third group of high-level test synthesis systems employs built-in self-test (BIST), specifically parallel BIST [7], [9]-[12], [26], [27]. Parallel BIST, which is based on random pattern testing, employ test pattern generators and test data evaluators for every module under test (which is usually a combinational circuit). Parallel BIST often achieves relatively high fault coverage compared with other BIST methods such as circular BIST [28]. In this paper, we present a high-level BIST synthesis method which employs the parallel BIST structure.

One of the earliest high level BIST synthesis methods based on the parallel BIST was proposed by Papachristou et al. [7]. In their method, all operations and variables are assigned to "testable functional block," which consists of input multiplexers, an ALU, and output registers. The objective of the assignment is to avoid self-adjacent registers (through which an input and the output of a module form a cycle) which are undesirable in BIST. Their method was later refined to further reduce the area overhead [26]. Avra proposed an elegant solution to avoid self-adjacent registers based on register conflict graphs [1]. Two variables of a data flow graph conflict if they are the input and output of the same module. The merger of the two variables results in a self-adjacent register and, therefore, should be avoided. Avra also suggested several schemes to reduce multiplexers and interconnections. The area overhead of Avra's method is less than that of Papachristou et al.'s earlier work [7] but more than that of their later work [26].

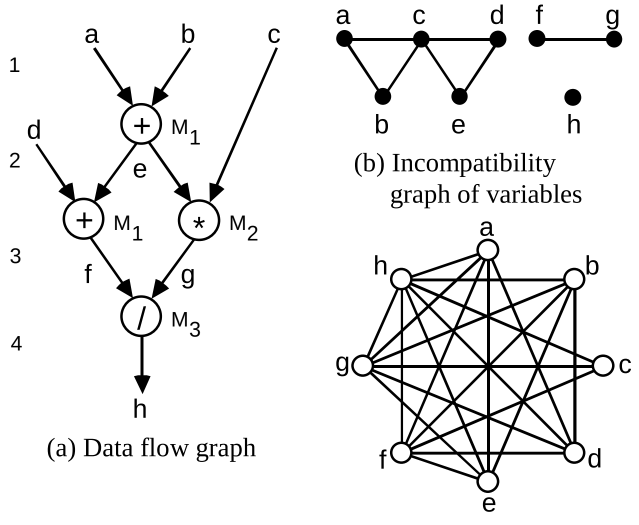
Parulkar et al. investigated a method which maximizes the sharing of test registers to reduce the area overhead [9]. During the register assignment phase, input and output variables of a data flow graph are merged to result in a maximal sharing of the registers and to avoid self-adjacent registers. A reverse perfect vertex elimination scheme is employed to obtain a maximal sharing of registers. Parulkar et al.'s method performs better (in terms of area overhead) than Avra's method [1] and Papachristou et al.'s later work [26].

All the above mentioned works focused on minimization of area overhead in BIST synthesis. For those methods, test time is not a concern in the design process, and is determined from the synthesized circuit through a post-process. In order to reduce test time in BIST, Harris and Orailoglu examined conditions which prevent concurrent testing of modules [4], [27]. They identified two types of conflicts; namely hardware conflict and software conflict. The synthesis process is guided to avoid such conflicts for the synthesized circuit. They reported that test time for example circuits is reduced (presumably at the cost of higher area overhead) through the proposed method [4], [27].

The existing high-level BIST synthesis methods described above focus on one objective, minimizing either the area overhead [1], [7], [9]-[12], [26] or the test time [4], [27]. Hence, those methods do not render exploration of large design space, which may result in a local optimum. Another aspect which was overlooked in those methods is that area overhead and test time are often traded in BIST (as well as other design-for-testability methods). Therefore, it is more desirable to offer various design alternatives (with different area overhead and test time) to the designer, and let the designer choose a proper design for his/her needs. Our method intends to address those two problems.

For a scheduled and module-assigned data flow graph, our method allocates signature registers which guarantee the circuit be tested in *k*-test session, where *k* is 1, 2, ..., N, and N is the number of modules. Our method tries to find an optimal design (which incurs the smallest area overhead) for each k-test session. Hence, it explores a far larger design space compared with other methods. It also allows designers to trade area and test time. A designer whose concern is only area overhead can choose the most area efficient design among N designs, while one concerned with only test time chooses the design for *k*=1.

This paper is organized in the following manner. In Section 2, we briefly explain BIST synthesis and describe necessary terms. In Section 3, we describe the proposed method. Guidelines for the merger of variables are explained step by step. Section 4 contains experimental results for our method. The performance of our method is compared with other BIST synthesis methods. Section 5 concludes the paper.



(a) Data flow graph

(b) Incompatibility graph of variables

(c) Compatibility graph of variables

Fig. 1. A data flow graph and its incompatibility and compatibility graphs

II. PRELIMINARIES

In this section, we explain high level BIST synthesis using an example data flow graph and describe necessary terms to understand our method. We also discuss an issue regarding self-adjacent registers.

2.1 High-Level Synthesis and Data Flow Graph

High-level synthesis is a process of transforming a behavioral description into a structural description comprising data path logic and control logic [29]. First, a behavioral description is converted into a control data flow graph. Then operations are scheduled in clock cycles (scheduling), a hardware module is assigned to each operation (module assignment), and registers are assigned to input and output variables (register assignment). Among the three operations, register assignment has the major impact on parallel BIST, and, hence, is the subject of this paper.

A data flow graph in which scheduling and module assignment have been completed is shown in Fig. 1(a). Shaded lines in the data flow graph denote clock cycle boundaries. An input or output variable on a clock boundary should be stored in a register. In other words, a register should be assigned to each input or output variable, which is called register assignment. If two variables overlap at a clock boundary, the two variables are *incompatible*, and two incompatible variables cannot share the same register. The incompatibility graph of input and output variables for the data flow graph is shown in Fig. 1(b). A vertex of the graph is a variable, and an edge exists between each pair of *incompatible* vertices. A compatibility graph of variables is readily derived from an incompatibility graph and is shown in Fig. 1(c). In this case, an edge between two vertices indicates that the two vertices are *compatible*.

If a variable v is an input (output) of a module M_i, the module is called the *destination* (*source*) *module* of variable v. A destination module does not exist for a primary output variable such as variable h in the data flow graph in Fig. 1(a). Similarly, the source module does not exist for a primary input variable. The source module of a variable v is denoted as SM(v). For example, SM(f) for the data flow graph is M_1, and SM(a)=ϕ. A variable whose life spans more than one clock boundary is called *delayed variable*. Variable c for the data flow graph is a delayed variable, as it spans two clock cycles. During the register assignment, a delayed variable spanning *n* clock cycles is often split into *n* variables to increase the flexibility of the register assignment [1].

A complete graph is one such that every pair of vertices has an edge. A clique of a graph is a complete subgraph. The size of a clique is the number of vertices of the clique. If a clique is not contained by any other clique, it is called maximum. For the compatibility graph in Fig. 1 (c), the graph {a,d,f,h} is a clique of size 4 and is maximum. A clique partition is to partition a graph into a disjoint set of cliques. *Maximum clique partition* is a clique partition with the smallest number of cliques. To find a maximum clique partition is an NP-complete problem [29].

A vertex v is called *common neighbor* of a subgraph provided vertex v is not contained in the subgraph and has an edge with every vertex of the subgraph. If a vertex v is not a common neighbor of a subgraph, but has an edge with at least one vertex of the subgraph, the edge is called *non-common edge*. For example, vertex h in the compatible graph in Fig. 1(c) is a common neighbor of a subgraph {c,d,f}, but vertex g is not. As g has an edge c-g with vertex c, the edge c-g is a non-common edge for the subgraph.

2.2 Register Assignment

When ignoring testability of a synthesized circuit, the goal of the register assignment is to assign the least number of registers for a given scheduled and module-assigned data flow graph. Optimal register assignment is a maximum clique partition problem for the compatibility graph of input and output variables. Suppose that a set of variables {v_1, v_2, v_3, ...} is assigned to register R_i. Register R_i can be considered as a set whose elements are v_1, v_2, v_3, ... , i.e., R_i={v_1, v_2, v_3,...}. We define the following two terms on register R_i.

Definition: The set of source modules for register R_i, denoted as SM(R_i), is the collection of source modules of the variables assigned to R_i, i.e., SM(R_i)={SM(v_1), SM(v_2), SM(v_3)...}. The number of source modules of R_i is |SM(R_i)|, where |S| is the cardinality of a set S.

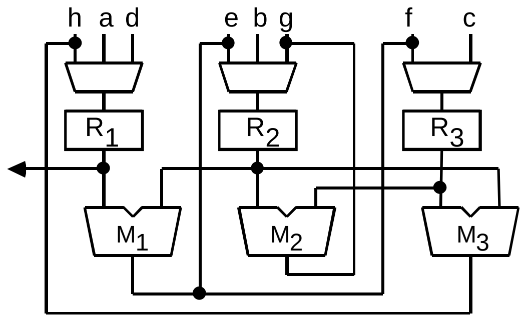


Fig. 2. Data path logic under the register assignment

Definition: The set of destination modules for register R_i, denoted as DM(R_i), is defined as {DM(v_1), DM(v_2), DM(v_3)...}. The number of destination modules for R_i is |DM(R_i)|.

The terms are illustrated for the following register assignments of the variables in Fig. 1(a). (Note that it is a minimum clique partition.)

R_1={a,d,h}, R_2={b,e,g}, and R_3={c,f}.

The resultant data path logic under the register assignment is given in Fig. 2. For the register assignment of R_2, SM(R_2)={SM(b), SM(e), SM(g)}={M_1, M_2}, and DM(R_2) ={DM(b), DM(e), DM(g)}={M_1, M_1, M_2, M_3}={M_1, M_2, M_3}.

2.3 BIST Synthesis and Test Registers

Parallel BIST needs to construct a test structure in which a test pattern generator and a signature register are connected to each input port and each output port of a module, respectively. All the test registers are reconfigured from system registers. *The objective of the BIST synthesis considered in this paper is to allocate registers to incur the least area overhead, while the circuit can be tested in k-test session*. After the allocation of registers, the parallel BIST structure is constructed through reconfiguration of registers into four different types of test registers to be described next.

A system register may be converted into one of four different types of test register: test pattern generator (TPG), multiple input signature register (short for signature register), built-in logic block observer (BILBO) [30], and Concurrent BILBO (CBILBO) [31]. If a test register behaves as a TPG in a test session and a signature register (SR) in another session, the test register should be reconfigured as BILBO. If a test register should be a TPG and a SR in the *same* test session, it should be reconfigured as a CBILBO. Reconfiguration of a register into CBILBO requires double the number of flip-flops of the register. Hence, it is expensive in hardware cost. CBILBOs are often required for self-adjacent registers.

In general, a TPG can be shared between modules, as long as each input of a module receives test patterns from a different TPG. (So that test patterns of different input ports are not correlated.) However, a SR cannot be shared between modules tested in the same

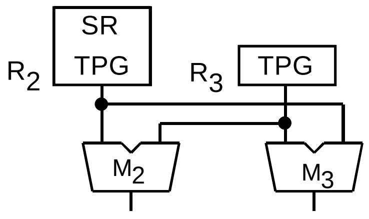


Fig. 3. Sharing of test pattern generators

session. One SR is necessary for each module tested in the same session. This implies that *the number of test sessions is determined by the allocation of SRs, not by TPGs*. Our method, to be presented in the next section, is based on this premise.

2.4 Self-adjacent Registers and Sharing of Signature Registers

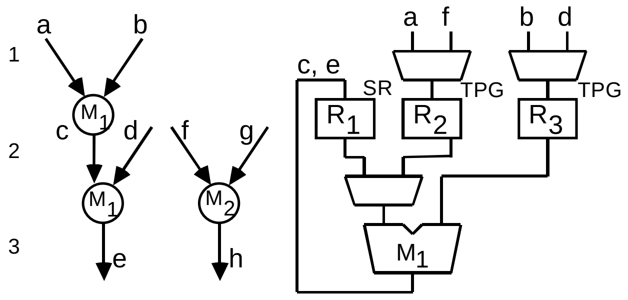
A register, through which a cycle is formed for a module, is called self-adjacent. Assignment of input and output variables of a module to the same register creates a self-adjacent register, viz. register R_2 in Fig. 2. The main focus of early high-level BIST synthesis methods focused on the avoidance of self-adjacent registers [1], [7]. However, self-adjacent registers do not necessarily require CBILBOs. Consider part of a data flow graph given in Fig. 4(a). The data path logic under a register assignment, R_1={c,e}, R_2={a,f}, and R_3={b,d}, is shown in Fig. 4(b). The necessary reconfiguration of registers to test module M_1 is also indicated in the figure. Note that Register R_1 is self-adjacent, but it is reconfigured to a SR, not a CBILBO.

Parulkar et al.'s method aims to reduce the overall area overhead by sharing registers in their maximum capacity [9]. However, excessive sharing of signature registers is unnecessary. In fact, it may result in higher area overhead. Consider the cases given in Fig. 5. Registers R_1 and R_2 are reconfigured into signature registers during testing. The sharing of R_1 with M_3 in Fig. 5(a) is unnecessary with 2-test session as shown in Fig. 5(b), and it creates an unnecessary path to incur higher area overhead. Hence, our method assigns one signature register to each module to avoid such a waste. Note that M_2 in Fig. 5(a) has two available SRs.

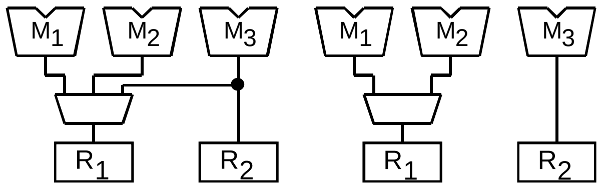
III. PROPOSED *k*-TEST SESSION METHOD

In general, the area overhead and the test time are traded in BIST. However, an approach which aims to achieve one objective, either minimal area overhead or shortest testing time, while ignoring the other one, may direct the design toward one extreme. As a result, the approach does not explore a wide range of design space to yield a non-optimal design.

In this section, we present our method which is intended to correct the problem. Our method is concerned with register assignment to embed the parallel BIST



(a) Data flow graph (b) BIST configuration for M_1
Fig. 4. Self-adjacent register and its reconfiguration to a SR



(a) Excessive sharing (b) Proper sharing
Fig. 5. Allocation of signature registers

structure for a given data flow graph *in which scheduling and module assignment have been completed*. To explore a large design space, our method tries to find an optimal register assignment for each *k*-test session, where *k* = 1, 2, ..., N. A register assignment for a k-test session guarantees that the synthesized circuit can be tested in *k*-test session. Therefore, our method offers a range of designs with different figures of merit in area and test time, and it enables a designer to select an appropriate design for his/her needs.

3.1 Overall approach

The key idea of our approach is to explore a large design space by considering all the possible test sessions. The upper bound of the number of possible test sessions is equal to the total number of modules, N, which is usually small enough to make our method computationally tractable. Our method tries to find an optimal register assignment (which incurs the least area overhead) for each test session *k*, where *k*=1,2, ..., N. To find an optimal register assignment for each test session is computationally intensive. As stated in Section 2.3, we explained that allocation of signature registers, not test pattern generators, determines the number of test sessions. Thus, we consider the allocation of only signature registers in the first phase. Our method merges output variables (which are candidates of signature registers) to achieve a k-test session. In the next phase, our method merges the remaining variables (input variables and unallocated output variables) to maximize sharing of the allocated signature registers with other registers. After the entire signature registers are allocated and possibly shared with other registers, our method identifies candidate registers for test pattern generators and optimizes the design to reduce multiplexers and interconnections.

Our approach, which allocates signature registers first to achieve k-test session and then shares the signature registers with other registers, makes the procedure computationally simple, yet effective (as indicated by our experimental results to be given Section IV).

3.2 Phase I: Allocation of Signature Registers

The given design for our method is a data flow graph in which the process for scheduling and module assignment has been finished. The task in Phase I is to allocate signature registers to modules, so that all the modules can be tested in *k*-test session. Let us suppose that there are N modules, M_1, M_2, and M_N, for the data flow graph under consideration. Let S_R be a set of signature registers for the circuit. Necessary conditions for an optimal assignment of signature registers to achieve *k*-test session are given below.

(i) $\bigcup_{R_i \in S_R} SM(R_i) = \{M_1, M_2, ...M_N\}$.

(ii) $\bigcap_{R_i \in S_R} SM(R_i) = \varnothing$.

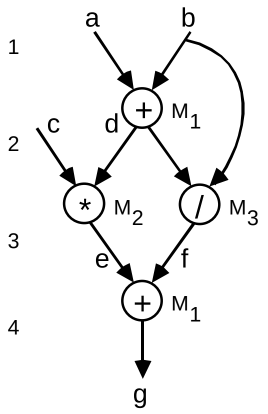
(iii) $\forall R_i \in S_R,\ |SM(R_i)| \le k$.

The first condition specifies the coverage of all the modules. The second one states only one signature register be assigned to each module. (Assignment of multiple signature registers to a module is not optimal.) All the source modules of a signature register R_i, i.e., $SM(R_i)$, share the same signature register R_i. Hence, they should be tested in different test sessions to necessitate the third condition.

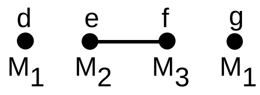
Allocation of signature registers is the process of merging output variables and of assigning registers to merged output variables, while meeting the three conditions described above. The goal of the allocation is to allocate the *least* number of signature registers with the *least* number of output variables assigned for each register. The intention of assigning the least number of output variables to each register is to increase the number of the unassigned output variables, which may be shared with other variables in Phase II. Thus, two output variables with the same source module should not be merged and assigned to the same register in this phase. The condition is specified in the following:

(iv) $\forall u,v \in R_i$, if $u \neq v$, $SM(u) \neq SM(v)$.

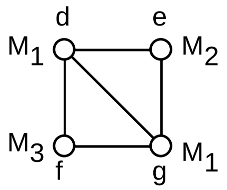
We use the compatibility graphs of output variables to identify candidate output variables for possible mergers. A clique of an output compatibility graph indicates that the variables of the clique can be merged. In order to achieve *k*-test session, the clique size of each clique should not exceed *k*. (Refer to condition (iii).) In addition, a clique should not include any two variables



(a) Data flow graph



(b) Incompatibility graph of the output variables



(c) Compatibility graph of the output variables

Fig. 6. A data flow graph with module assignment

with an identical source module. (Refer to condition (iv).) We illustrate the process using an example data flow graph.

Consider the data flow graph given in Fig. 6(a). The three operators, +, *, and /, are binded to M_1, M_2 and M_3, respectively. The incompatibility and compatibility graphs of output variables and their associated source modules are shown in Fig. 6(b) and Fig. 6(c), respectively.

From the compatibility graph of the output variables in Fig. 6(c), we look for cliques with size *k* for a *k*-test session, but any two vertices of a clique should not have the same module (refer to condition (iv)). For *k*=1, each vertex is a clique. Hence, there are four cliques, {d}, {e}, {f}, and {g}. For a 2-test session, i.e., *k*=2, there are {d,e}, {d,f}, {e,g}, and {f,g}. When *k* is three, there is no such clique. Note that the cliques {d,e,g} and {d,f,g} have two vertices, d and g, whose source module is the same. This means that a 3-test session is not optimal for the circuit. This can readily be seen from the incompatibility graph in Fig. 6(b). The maximum clique size of the graph is two. Hence, there are at least two signature registers available for the circuit. Clearly, testing three modules in three test sessions is not optimal when two signature registers are available.

If there are multiple candidate cliques for a *k*-test session, we choose a clique. Then the chosen clique and all the edges connected to the clique are removed from the graph. All the vertices (i.e., nodes) whose source modules have been covered by the selected clique and the associated edges of the vertices are also removed. We repeat the procedure for the resultant graph until all the modules are covered. It should be noted that if a clique of size *k* is not present for a resultant graph, a clique with the next candidate clique should be chosen. This procedure is illustrated in the following.

For the previous example with *k*=2, all four cliques, {d,e}, {d,f}, {e,g}, and {f,g}, have the same number (one) of common neighbors. Suppose we choose

Table 1. Possible optimal mergers for the data flow graph in Fig. 6(a)

k	Merger I	Merger II	Merger III	Merger IV
1	d e f	e f g		
2	d,e f	d,f e	e,g f	f,g e

clique {d,e}. The clique and all the associated edges, d-f, d-g, and e-g, are removed. As module M_1 is covered by clique {d,e}, we also remove vertex g and its associated edge, f-g. Thus, the resultant graph has only vertex f. Vertex f is selected to cover M_3 in the next process. All possible mergers for the data graph are given in Table 1.

The output variables of a group[1] (denoted as a circle) in Table 1 are merged (possibly with other input and output variables in Phase II) and assigned to a signature register. The number of groups for a merger is the number of signature registers to be allocated. For example, a merger {d,e} and {f} for *k*=2 results in the allocation of two signature registers. The source modules of the variables in a group are tested sequentially using the same register. However, they can be tested simultaneously with source modules of the variables belonging to a different group. For example, source module M_1 of variable d and M_2 of variable e for Merger I under *k*=2 are tested sequentially, but either M_1 or M_2 can be tested simultaneously with module M_3 of variable f.

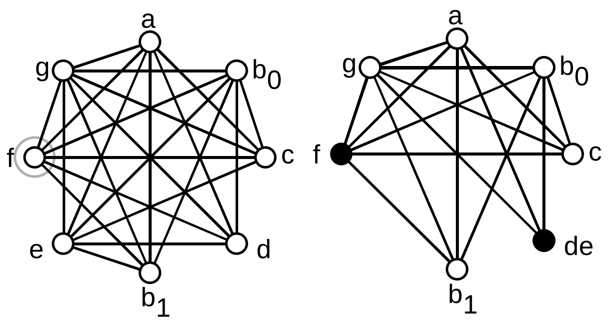
3.3 Phase II: Sharing of Signature Registers with Other Registers

In Phase II, we merge the output variables identified in Phase I with the remaining compatibility graph of input and output variables of the data flow graph. Two steps are involved in Phase II. First, all output variables of each group identified in Phase I are merged. Next, the merged output variables are further merged with other variables. When two compatible variables (i.e., vertices), X and Y, are merged, all the non-common edges of the subgraph {X,Y} should be deleted. This is true for mergers of more than two variables.

3.3.1 Merger of output variables

This process starts with merging all variables of each group identified in Phase I. All non-common edges of each group are deleted. In order to intact a signature register assigned to each group, output variables of a group should not be merged with the variables of another group. It necessitates removal of all the edges between the groups.

We illustrate the process using the data flow graph in Fig. 6(a). Suppose that we have chosen Merger I for *k*=2 in Table 1. The groups of output vertices to be merged are {d,e} and {f}. (In fact, group {f} has one



(a) Before the merger (b) After the merger

Fig. 7. Compatibility graphs before and after the merger

vertex and no need for merging.) The compatibility graph of the data flow is shown in Fig. 7(a). Note that delayed variable b is split into two variables, b_0 and b_1. As vertices d and e are merged, three non-common edges, c-e, b_1-e, and f-d, are removed. Then all the edges between the two groups {d,e} and {f} are deleted. Edge d-f is deleted (In fact, it has already been deleted.). The resultant compatibility graph of variables is shown in Fig. 7(b). The filled circles in the figure denote variables assigned to signature registers.

3.3.2 Merger of the remaining variables

After the output variables for a k-test session are merged, all the remaining compatible vertices (i.e., input and output variables) are to be merged. The goal of the merger is to minimize the number of resultant vertices in the graph, equivalently the number of allocated registers. The process is minimum clique partition. A minimum clique partition is NP-complete, and, hence, a heuristic is necessary for the process. In our case, if there are multiple minimum clique partitions, we need to select a partition which is likely to incur the least area overhead. (Note that all partitions require *k*-test session.) In the following, we describe guidelines which aim to lead the process toward a minimum clique partition and for the least area overhead.

Our method is to merge a pair of compatible vertices iteratively until no further merger is possible. The guidelines to choose a pair of compatible vertices to be merged are given in order of descending priority. A guideline *i* is applied only when its immediate guideline *i*-1 (equivalently all the previous guidelines) is tied. The guidelines are illustrated for the compatibility graph in Fig. 6(b). The first two guidelines aim to reduce the total number of registers for the circuit by sharing as many

[1] Group and clique are the same and are used interchangeably.

variables as possible. The rest of the guidelines are for the reduction of test registers and/or associated hardware, specifically multiplexers and interconnections.

Guideline 1: Choose a pair with the largest number of common neighbors.

This guideline aims to merge a pair of vertices which belong to a maximum clique of the graph. As an illustration, the number of common neighbors for each compatible pair for the compatibility graph Fig. 7(b) is given in Table 2.

The largest number of common neighbors are four for three pairs of vertices, {a,g}, {b_0,g} and {f,g}. The three pairs of vertices belong to maximum cliques, {a,c,f,g}, {a,b_1,f,g} and {b_0,c,f,g} of the graph. As the three pairs are tied, guideline 2 is used to break the tie.

Guideline 2: Choose a pair with the least number of non-common edges.

When a pair of vertices are merged, all the non-common edges of the pair of vertices are deleted. Guideline 2 selects a pair whose merger deletes the least number of edges from the graph. This means that the merger of the two vertices minimally decreases the chances for further mergers. In the example graph, the merger of {a,g} deletes edge b_0-g, the merger of {b_0,g} deletes edge a-g and the merger of {f,g} deletes edge g-de. All three mergers delete only one vertex. Hence, guideline 3 is used to break the tie.

Guideline 3: Choose a pair which is split from the same variable.

As described in Section 2.4, if a variable spans a number of control steps, then it is split into each control step to yield a more flexible register assignment at the cost of additional multiplexers. At this stage, the split of a variable is not useful. Hence, it is merged back to the same variable. For the three candidate pairs, no two vertices of a pair are split from the same variable.

Guideline 4: Choose a pair based on the number of destination modules.

This guideline is concerned with test pattern generators. First, choose a pair such that neither of its vertices is assigned to a signature register. Among multiple such pairs, choose a pair with the *largest* number of destination modules. This guideline aims to reduce the number of test pattern generators (TPGs) through the sharing of a TPG with multiple modules.

Second, if a variable of every pair is assigned to a signature register, choose a pair with the *smallest* number of destination modules. This guideline tries to minimize the likelihood of the register to be reconfigured as BILBO or CBILBO.

Table 2: The number of common neighbors for the graph in Fig. 7(b)

Compatible vertex pair	Common neighbors	No. of common neighbors
{a,c}	f, g	2
{a, de}	g	1
{a,b_1}	f, g	2
{a,f}	c, b_1, g	3
{a,g}	c, de, b_1, f	4
{b_0,g}	c, de, b_1, f	4
...	...	...
{f,g}	a, b_0, c, b_1	4
...	...	...

In the previous example, only two pairs {a,g} and {b_0,g} include variables not assigned to a signature register. Since the number of destination modules for each pair is one, i.e., |DM(a,g)| = |DM(b_0,g)| = 1, the next guideline is applied.

Guideline 5: Choose a pair with the smallest number of source modules.

Since a signature register is assigned to each module in Phase I, there is no need for further sharing of output registers at this stage. This guideline aims to reduce the number of multiplexers by reducing the sharing of output registers. In the previous example, both the pairs, {a,g} and {b_0,g}, have one source module, M_3. Hence, vertex pair {a,g} is chosen arbitrarily and merged. The resultant graph is shown in Fig. 8. The above procedure repeats for the resultant graph to merge the next pair of vertices.

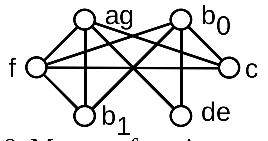


Fig. 8. Merger of vertices a and g

3.4 Post processing

After all the registers are assigned through Phase I and Phase II, we identify TPGs. It is always possible to allocate TPGs for each module by reconfiguring assigned registers or by sharing existing TPGs between different modules. (Note that a TPG is not shared between two inputs of the same module to avoid low fault coverage.) A signature register is converted into a BILBO if it should be a TPG in a different test session, or into a CBILBO if it should be a TPG in the same test session.

When all TPGs are allocated, we try to minimize multiplexers and interconnections using the methods suggested in [1] and [32]. The methods include exchanging input ports and rearranging multiplexers.

IV. EXPERIMENTAL RESULTS

In this section, we present experimental results on the performance of our proposed method. We also compare the performance of our method with two other BIST synthesis methods, RALLOC [1] proposed by Avra and BITS [9] proposed by Parulkar et al. We implemented all three BIST synthesis methods (our method called ADVAN, RALLOC, and BITS) in the C++ language. For the implementation of RALLOC and BITS, we faithfully followed the algorithms presented in [1] and [9].

4.1 Background

We measured the performance of the three BIST synthesis systems for six data flow graphs. The data flow graphs include the ones studied by Tseng and Siewiorek [32] called tseng and by Paulin and Knight [33] called paulin. The other four data flow graphs are a 6th order FIR (finite impulse response) filter, a 3rd order IIR (infinite impulse response) filter, a 4-point DCT (discrete cosine transformation) circuit, and a 4-tap wavelet filter. We adopted the scheduling and module assignment from [1] for tseng and paulin. The other four data flow graphs were synthesized using HYPER [34]. The width of the data path logic is eight for all the circuits. A detailed description of the circuits is available in [35].

In this paper, the area of a circuit is represented by the transistor count of registers and multiplexers in the circuit. *Data path logic of a circuit is not considered in the transistor count.* The number of transistors in test registers and multiplexers is based on the circuits of [30], [31] and is given in Table 3. In the table, #Trs and #MuxIn denote the number of transistors and the number of multiplexer inputs, respectively.

The reference circuit of a data flow graph, which was used to measure the area overhead of a BIST design, was obtained as follows. We collected all the circuits generated by HYPER and by the three BIST synthesis systems (ADVAN, RALLOC, and BITS). After test registers of BIST circuits were configured back to normal registers, we measured the area of individual circuits and chose the circuit with the least area as the reference circuit.

4.2. Experimental Results

The first experiment was to measure the performance of the proposed method and to examine the relationship between area and test time in high-level BIST synthesis. In allocating signature registers for our method, we considered *all possible optimal mergers* of the circuit for each test session. (Refer to Section 3.2 for optimal mergers.) Among all the mergers considered, we chose a merger which incurs the least area overhead.

Experimental results on area and test time (in terms of the number of test sessions) of synthesized BIST circuits are given in Table 4. The first row for each circuit entry is the reference circuit. Column headings for the table are explained below.

Table 3. Number of transistors of 8-bit test registers and multiplexers

a) Test registers

Type	Reg.	TPG	SR	BILBO	CBILBO
#Trs	208	256	304	388	596

b) Multiplexers

#MuxIn	2	3	4	5	6	7
#Trs	80	176	208	300	320	350

R	: the total number of registers
T	: the total number of test pattern generators
S	: the total number of signature registers
B	: the total number of BILBOs
C	: the total number of CBILBOs
M	: the total number of inputs of multiplexers
Area	: the number of transistors of the registers and the multiplexers
OH	: the area overhead of the BIST design (%)

From the table, the area overhead of ADVAN ranges from 8 percent to 40 percent. The area overhead of the last four circuits is less than 21 percent. ADVAN incurs relatively high area overhead for tseng and paulin due to the employment of CBILBOs and/or a large number of multiplexer inputs. However, since the area overhead of a circuit is computed without considering the area for the data path logic modules, the actual area overhead will be much lower than the ones presented in the table. For example, the area overhead of paulin is 40.8 percent for k=1. Circuit paulin contains two 8x8 multipliers, one 8-bit adder and one 8-bit subtractor. If the adder and the subtractor based on ripple carry propagation and array multipliers are used to implement the circuit, the actual area overhead would be reduced to about 18 percent. Therefore, it can be said that the area overhead of ADVAN is small to moderate for all the circuits tested.

The table shows that, in general, the area and test time are traded in high-level BIST synthesis. An illustrative example is fir6, in which the area overhead reduces monotically with an increase in the number of test sessions. When the number of test sessions, k, increases from 1 to 2, there is a substantial reduction in the area overhead for most circuits. However, an increase of k beyond 2 is little help in reducing the area overhead. In fact, the area overhead increases for some circuits. Therefore, it is a good idea to avoid testing an entire circuit in one test session if the area overhead is a major concern.

Next, we compare the performance of ADVAN with two other BIST synthesis systems, RALLOC [1] and BITS [9]. In order to make the comparison meaningful, the six data flow graphs used in the experiment employed the same scheduling and the same module assignment for all three systems. The performance of the three high-level

Table 4. Performance of ADVAN

Ckt	*k*	R	T	S	B	C	M	Area	OH(%)
tseng		5					15	3324	
	1	5	2	2	0	1	21	4240	35.1
	2	5	3	2	0	0	21	3944	26.8
	3	5	2	1	0	0	23	3996	28.4
paulin		5					17	1776	
	1	5	1	2	0	2	21	3000	40.8
	2	5	2	0	0	1	24	2580	31.2
	3	5	2	1	0	1	21	2564	30.7
	4	5	3	1	0	0	26	2588	31.4
fir6		7					24	2638	
	1	7	1	3	0	0	29	3372	21.8
	2	7	2	1	1	0	25	3200	17.6
	3	7	1	1	0	0	28	3072	14.1
iir3		6					27	2592	
	1	6	2	2	0	1	27	3268	20.7
	2	6	1	2	0	0	30	3056	15.2
	3	6	3	1	0	0	32	3136	17.3
dct4		6					34	2928	
	1	6	2	3	0	0	34	3312	11.6
	2	6	2	2	0	0	36	3344	12.4
	3	6	2	1	0	0	36	3200	8.5
	4	6	3	1	0	0	35	3324	11.9
wave-let4		7					45	3324	
	1	7	2	2	0	0	47	3720	10.6
	2	7	1	1	1	0	45	3648	8.9
	3	7	2	1	0	0	46	3612	8.0

Table 5. Performance of various high level BIST synthesis systems

Ckt	Method	R	T	S	B	C	M	Area	OH(%)
tseng (3)	Ref.	5					15	3324	
	ADVAN	5	2	1	0	0	23	3996	28.4
	RALLOC	5	1	0	3	0	14	3928	26.3
	BITS	5	2	1	1	0	20	4064	30.4
paulin (4)	Ref.	5					17	1776	
	ADVAN	5	3	1	0	0	26	2588	31.4
	RALLOC	5	1	0	3	0	25	2796	36.5
	BITS	5	2	0	0	1	27	2928	39.3
fir6 (3)	Ref.	7					24	2638	
	ADVAN	7	1	1	0	0	28	3072	14.1
	RALLOC	8	1	1	2	0	36	4180	36.9
	BITS	7	1	0	0	1	24	3156	16.4
iir3 (3)	Ref.	6					27	2592	
	ADVAN	6	3	1	0	0	32	3136	17.3
	RALLOC	7	1	0	2	0	38	4120	37.1
	BITS	6	2	0	2	0	29	3176	18.4
dct4 (4)	Ref.	6					34	2928	
	ADVAN	6	3	1	0	0	35	3324	11.9
	RALLOC	6	1	1	2	0	37	3716	21.2
	BITS	7	1	1	0	1	38	4180	30.0
wave -let4 (3)	Ref.	7					45	3324	
	ADVAN	7	2	1	0	0	46	3612	8.0
	RALLOC	8	1	0	3	0	50	5186	35.9
	BITS	7	1	0	2	0	40	3850	13.7

BIST synthesis systems is presented in Table 5. The table shows the case in which the number of test sessions is maximal for a given circuit. The number of test sessions for a circuit is given under the circuit name in the table.

From the table, ADVAN performs better than the other two systems in area for all circuits except tseng. For some circuits, area overhead of ADVAN is substantially less than that for another method. For example, the area overhead of ADVAN is 8 percent for wavelet4, while that for RALLOC is 36 percent. It should be pointed out that ADVAN does not add any additional registers. In other words, a reference circuit and its BIST circuit by ADVAN have the same number of registers. However, RALLOC needs one additional register for fir6, iir3, and wavelet, while BITS requires one additional register for dct4. The addition of registers incurs large area overhead as can be seen in Table 5. None of the three BIST synthesis systems requires additional registers for tseng and paulin. This is because that design space[2] for the two circuits is smaller than the other four circuits. Hence, there is not much freedom in the BIST design for the two circuits.

Each system has a different emphasis in register allocation to yield a different BIST design style. ADVAN tries to avoid reconfiguration of signature registers into TPGs. (Refer to Guideline 4 in Section 3.2.2.) As a result, none of the SRs are reconfigured to a TPG for ADVAN as shown in Table 5, and, hence, there is no BILBO or CBILBO for ADVAN. However, the cost is a large number of TPGs. The main focus of RALLOC is to avoid self-adjacent registers, which often require CBILBOs. So RALLOC does not have any CBILBOs, but many BILBOs. BITS intends to maximize the sharing of test resources, which leads to a small number of TPGs and SRs combined, but has CBILBOs. Based on this observation, it can be said that emphasis on one or a few strategies does not lead to an optimal BIST design. Instead a careful balance between different strategies is necessary for an optimal design.

In summary, the area overhead of ADVAN is moderate for the six circuits tested. ADVAN performs better in area overhead for most circuits than the other two BIST synthesis systems.

[2] The design space of a data flow graph may be represented as the ratio of (the number of variables + the number of nodes in the data flow graph) to (the number of modules + the number of registers in the data path logic).

V. SUMMARY

Existing high-level BIST synthesis methods focus on one objective, minimizing either the area overhead [1], [9]-[11], [24] or the test time [4], [27]. Hence, those methods do not render exploration of large design space, which may result in a local optimum. Another aspect which was overlooked in existing methods is the fact that area overhead and test time are often traded in BIST. Therefore, it is more desirable to offer various design alternatives (with different area overhead and test time) to the designer, and let the designer choose a proper design for his/her needs.

In this paper, we presented a method which is intended to correct the above two problems. To explore a large design space, our method tries to find an optimal register assignment for each k-test session, where $k = 1, 2, \ldots N$, and N is the number of modules. A register assignment for a k-test session guarantees that the synthesized circuit can be tested in *k*-test session. Therefore, our method offers a range of designs with different figures of merit in area and test time.

Our experimental results show that the area overhead of our method called ADVAN is moderate for the six circuits tested. ADVAN performs better in area overhead for most circuits than two other BIST synthesis systems, RALLOC [1] and BITS [9]. Overall, the distinct advantage of ADVAN over other existing methods is that our method offers various design alternatives to the designer, so it enables a designer to select an appropriate design for his/her needs.

REFERENCES

[1] L.J. Avra, "Allocation and Assignment in High-Level Synthesis for Self-Testable Data Paths," *Proc. Int. Test Conf.*, pp. 463-472, Oct. 1991.

[2] C.-H. Chen, T. Karnik and D.G. Saab, "Structure and Behavioral Synthesis for Testability Techniques," *IEEE Trans. on Computer-Aided Design*, Vol. 13, No. 6, pp 777-785, June 1994.

[3] S. Chiu and C.A. Papachristou, "A Design for Testability Scheme with Applications to Data Path Synthesis," *Proc. 28th Design Automation Conf.*, pp. 271-277, June 1991.

[4] I.G. Harris and A. Orailoglu, "Microarchitactural Synthesis of VLSI Designs with High Test Concurrency," *Proc. 31st Design Automation Conf.*, pp. 206-211, June 1994.

[5] F.F. Hsu, E.M. Rudnick and J.H. Patel, "Enhancing High-Level Control-Flow for Improved Testability," *Intl. Conf. on Computer-Aided Design*, Nov. 1996.

[6] A. Majumdar, R. Jain, and K. Saluja, "Incorporating Testability Considerations in High-Level Synthesis," *J. Electronic Testing: Theory & Applications*, pp. 43-55, Feb. 1994.

[7] C.A. Papachristou, S. Chiu and H. Harmanani, "A Data Path Synthesis Method for Self-Testable Designs, " *Proc. 28rd Design Automation Conf.*, pp. 378-384, June 1991.

[8] C.A. Papachristou and J. Carletta, "Test Synthesis in the Behavioral Domain," *Proc. Int'l. Test Conf.*, pp. 693-702, Oct. 1995.

[9] I. Parulkar, S. Gupta, and M.A. Breuer, "Data Path Allocation for Synthesizing RTL Designs with Low BIST Area Overhead," *Proc. 32nd Design Automation Conf.*, pp. 395-401, June 1995.

[10] I. Parulkar, S. Gupta, and M.A. Breuer, "Introducing Redundant Computations in a Behavior for Reducing BIST Resources," *Proc. 35th Design Automation Conf.*, pp. 548-553, June 1998.

[11] I. Parulkar, S. Gupta, and M.A. Breuer, "Scheduling and Module Assignment for Reducing BIST Resources," *Proc. DATE*, pp. 66-73, Feb. 1998.

[12] I. Parulkar, S.K. Gupta and M.A. Breuer, "Lower Bounds on Test Resources for Scheduled Data Flow Graphs," *Proc. 33rd Design Automation Conf.*, pp. 143-148, June 1996.

[13] D. Gajski, N. Dutt, A. Wu, and S. Lin, High-Level Synthesis: Introduction to Chip and System Design, Kluwer Academic Publishers, 1992.

[14] L.J. Avra and E.J. McCluskey, "High-Level Synthesis of Testable Designs: an Overview of University Systems," *Proc. Int'l. Test Conf.*, TS Paper 1.1, pp. 1-8, Oct. 1994.

[15] P. Vishakantaiah, J.A. Abraham, and M. Abadir, "Automatic Test Knowledge Ext:raction from VHDL (ATKET)," *Proc. 29th Design Automation Conf.*, pp. 273-278, June 1992.

[16] P. Vishakantaiah, T. Thomas, J.A. Abraham, and M. Abadir, "AMBIANT-Automatic Generation of Behavioral Modifications for Testability," *Intl. Conf. on Computer Design*, pp. 63-66, Oct. 1993.

[17] C.-H. Chen and D.G. Saab, "A Novel Behavioral Testability Measure," *IEEE Trans. on Computer-Aided Design*, Vol. 12, No. 12, pp. 1960-1970, Dec. 1993.

[18] T.-C. Lee, N.K. Jha, and W.H. Wolf, "Behavioral Synthesis for Highly Testable Data Paths under the Non-Scan and Partial Scan Environrnents," , *Proc. 30th Design Automation Conf.*, pp.292-297, June 1993.

[19] T.-C. Lee, N.K. Jha, and W.H. Wolf, "A Conditional Resource Sharing Method for Behavioral Synthesis of Highly Testable Data Paths," *Proc. Int'l. Test Conf.*, pp.744-753, Oct. 1993.

[20] T. Kim, K.-S. Chung, and C.L. Liu, "A Stepwise Refinement Data Path Synthesis Procedure for Easy Testability," *European Test Conf.*, pp.586-590, Mar. 1994.

[21] S. Bhatia and N.K. Jha, "Genesis: A Behavioral Synthesis System for Hierarchical Testability," *European Test Conference*, pp.272-276, Mar. 1994.

[22] A. Majumdar, R. Jain, and K. Saluja, "Behavioral Synthesis of Testable Designs," *24th Fault-Tolerant Computing Symposium*, pp.436-445, June 1994.

[23] S. Bhatia and N.K. Jha, "Behavioral Synthesis for Hierarchical Testability of Controller/Data Path Circuits with Conditional Branches," *Intl. Conf. on Computer Design*, Oct. 1994.

[24] T.-C. Lee, W.H. WoIf, N.K. Jha and J.M. Acken, "Behavioral Synthesis for Easy Testability in Data Path Allocation," *Intl. Conf. on Computer Design*, pp.29-32, Oct. 1992.

[25] T.-C. Lee, W.H. Wolf, and N.K. Jha, "Behavioral Synthesis for Easy Testability in Data Path Scheduling," *Intl. Conf. on Computer-Aided Design*, pp.616-619, Nov. 1992.

[26] H. Harmanani and C.A. Papachristou, "An Improved Method for RTL Synthesis with Testability Tradeoff," *Intl. Conf. on Computer-Aided Design*, pp. 30-35, Nov. 1993.

[27] A. Orailoglu and I.G. Harris, "Microarchitectural Synthesis for Rapid BIST Testing," *IEEE Trans. Computer-Aided Design*, Vol.16, No. 6, pp. 573-586, June 1997.

[28] S. Pilarski, A. Krasniewski, and T. Kameda, "Estimating Testing Effectiveness of the Circular Self-Test path Technique," *IEEE Trans. on Computer-Aided Design*, Vol. 11, No. 10, 1301-1316, Oct. 1992.

[29] G. DeMichelli, Synthesis and Optimization of Digital Circuits, McGraw Hill, 1994.

[30] Konemann, B.J. Mucha, and G. Zwiehoff, "Built-In Logic Block Observation Techniques," *Proc. Int'l Test Conf.*, pp. 37-41, Oct. 1979.

[31] L.-T. Wang and E.J. McCluskey, "Concurrent Built-In Logic Block Observer (CBILBO)," *Int. Symp. On Circuits and Systems*, pp. 1054-1057, May 1986.

[32] C. Tseng and D.P. Siewiorek, "Automated Synthesis of Data Paths in Digital Systems," *IEEE Trans. on Computer-Aided Design*, pp. 379-395, July 1986.

[33] P.G. Paulin and J.P. Knight, "Force-directed Scheduling for the Behavioral Synthesis of ASICs," *IEEE Trans. on Computer-Aided Design*, Vol. 8, No. 6, pp. 661-679, June 1989.

[34] M. Potkonjak and J. Rabaey, "A Scheduling and Resource Allocation Algorithm for Hierarchical Signal Folw Graphs," *Proc. 36th Design Automation Conf.*, pp. 7-12, June 1989.

[35] T. Takahashi, H.B. Kim, and D.S. Ha, "BIST Synthesis - I," Technical Report, VISC-DSH-1-98, Department of Electrical and Computer Engineering, Virginia Polytechnic Institute and State University, July 1998.