Data Path Allocation for Synthesizing RTL Designs with Low BIST Area Overhead*

Ishwar Parulkar, Sandeep Gupta and Melvin A. Breuer
Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, CA 90089-2562.

Abstract—**Built-in self-test (BIST) techniques have evolved as cost-effective techniques for testing digital circuits. These techniques add test circuitry to the chip such that the chip has the capability to test itself. A prime concern in using BIST is the area overhead due to the modification of normal registers to be test registers. This paper presents data path allocation algorithms that 1) maximize the sharing of test registers resulting in a fewer number of registers being modified for BIST, and 2) minimize the number of CBILBO registers.**

I. Introduction

Built-in self-test (BIST) techniques have gained acceptance for testing complex digital designs. These techniques involve modification of the hardware on the chip such that the chip has the capability to test itself. One of the considerations in BIST techniques is the extra area needed for the test circuitry. How to reduce the BIST area overhead without sacrificing the quality of the test is an important research problem [1]. Considering testability at an earlier stage in a design can lead to a more efficient exploration of the design space, thus resulting in a circuit that requires minimal BIST area overhead and that meets area, throughput and other requirements. The field of high-level synthesis has made significant progress in addressing area-performance requirements. However investigation of synthesis methods that take into account testability has only recently received attention from the research community [2],[3],[4]. The objective of these methods is ease of test generation and efficient partial scan. For designs that support BIST, Avra proposed a register allocation method that minimizes the number of self-adjacent registers in the design [5]. The assumption in this work is that every self-adjacent register needs to be modified to be a CBILBO register, and thus the overhead is high. Papachristou et al. first presented a combined register and ALU allocation method that generates self-testable designs that do not have any self-loops [6]. The approach is based on constraining the allocation to generate a self-testable template which is very restrictive and hence results in exploring a small subspace of the testable design space. Later they presented a method of generating self-testable designs by extending the self-testable template to include one specific configuration of a self-adjacent register [7]. This approach is still restrictive because it does not consider many other self-adjacency scenarios that do not require CBILBOs and is based on synthesizing templates by simultaneous constrained allocation of ALUs, registers and interconnect.

Given a scheduled data flow graph we allocate operations, variables and data transfers such that the functional constraints are satisfied and the area overhead required for BIST is minimal. The BIST methodology used is based on the concept of an I-path [8] and the sharing of I-paths between modules. We show that self-adjacency does not necessarily imply poor testability and have derived *exact* conditions of register assignment which necessitate CBILBOs. Our model subsumes the testability design styles considered in [5] and [7]. The conditions derived and the *separate* assignment of operations, variables and data transfers enables an efficient exploration of a larger testable design solution space.

II. Test methodology

The allocation scheme presented in this paper is directed towards synthesizing data paths that are to be tested using a partial intrusion pseudo-random BIST methodology. In this methodology, some of the registers in the data path are reconfigured to support test pattern generation (TPG) and signature analysis (SA) during the test mode. A built-in logic-block observation (BILBO) register is a design capable of these modes. The basic BILBO BIST architecture consists of partitioning a circuit into a set of registers and combinational blocks. For complete testing of all the combinational blocks in the design, different mappings of registers to TPGs and SAs is required. The concept of an I-path can be used effectively to explore the various mappings [8]. The data path architectures comprising registers and combinational operator modules with the multiplexer connectivity model lend themselves naturally to the concept of I-paths and the BILBO BIST methodology. Also the mapping of registers to TPGs and SAs is independent of the function and the gate-level implementation of the operator modules. Hence the testability constraints for this BILBO BIST methodology are appropriate for consideration during high-level synthesis.

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

32nd ACM/IEEE Design Automation Conference ®

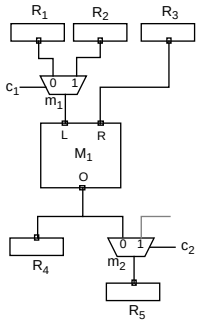


Fig. 1. A generic configuration with simple I-paths

Definition 1: (Abadir and Breuer [8]) An identity path, **I-path** is a data path from a primary input or a register to an input port of an operator module or from an output port of an operator module to a primary output or a register so that data can be transferred unaltered. The first and the last elements of an I-path are called the **head** and **tail**, respectively, of the I-path. A **simple I-path** is an I-path consisting of at most one register and no operator modules.

An example of a binary operator module M_1 and simple I-paths to and from its ports is shown in Fig. 1. Input port R has a simple I-path from R_3. This I-path is active at all times and is the only I-path to port R. Input port L has simple I-paths from R_1 and R_2 that pass through multiplexer m_1. I-paths with multiplexers can be activated by appropriate control signals, for example c_1 in this case.

A configuration of I-paths that covers all the ports of a module is called a BIST **embedding** of the module. The heads of the I-paths to the input ports are modified as TPGs and the tails of the I-paths from the output ports are modified as SAs. If the configuration chosen is such that a head and a tail are the same register then that register has to act as a TPG and SA at the same time. To ensure high fault-coverage a concurrent built-in logic block observation (CBILBO) register is required [9].

Generally, there is more than one embedding for a module. The choice of TPGs and SAs for testing this module largely depends on how and which registers are connected to the rest of the modules. For minimizing the BIST area overhead, the design is analyzed globally to determine the test resource allocation such that all operator modules are tested with a minimal number of registers modified as test resources. Also a CBILBO register has an area approximately twice that of a normal register and hence another objective is to minimize the number of CBILBOs. Since minimal area overhead is our objective, it is not necessary to test all the combinational modules at the same time, i.e. in one test session.

The assignment of registers and interconnect presented in this paper is done with the aim of maximizing sharing of I-paths between different modules. Also the assignment procedure takes into account the possibility of requiring CBILBO registers in the testable design and avoids such assignments. This leads to superior results compared to existing synthesis procedures.

III. Module and register assignment

The behavioral description is assumed to be given in the form of a data flow graph (DFG) $G = (V, E)$ where

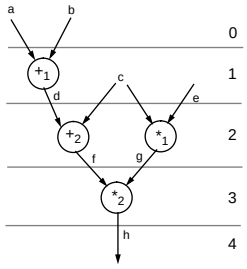


Fig. 2. A scheduled DFG

V is the set of operations and E is the set of variables (operands and results of the operations) and a schedule $S : V \rightarrow \{ 1, 2, 3,... \}$ where $S(v)$ corresponds to the control step in which operation v is scheduled. All operators are assumed to be binary and commutative. Non-commutative operators can be handled by adding additional constraints in our assignment procedures. Unary operators can be treated as a special case of binary operators.

The assignment is performed in the following order - module assignment, register assignment and finally interconnect assignment. We chose the above order for the following reasons. The module assignment space is relatively much smaller than the other spaces and the impact of module assignment on the *functional area* is large. Hence there is little flexibility within the module assignment solution space for improving testability. Most of the flexibility for minimizing test resources for BIST exists in register assignment. Also, registers can be viewed as *potential* test resources only *after* the module assignment is fixed. Interconnect assignment is strongly related to module and register assignment and its effect on functional area can be implicitly considered during those phases. In our approach, the interconnect assignment that follows register assignment tries to make the best use of the register assignment to further reduce BIST overhead.

Module assignment is done without any testability consideration. Existing algorithms that optimize area are used. The module assignment is defined as $\sigma : V \rightarrow M$ where M is the set of available modules. The subset of V mapped onto module M_i will be referred to as V_i. Each operation $v \in V_i$ will be referred to as an **instance** of M_i.

Definition 2: The **temporal multiplicity** of module M_i, $TM(M_i)$ is the number of operations from V mapped onto M_i, i.e. $TM(M_i) = \mid V_i \mid$.

Consider the scheduled DFG shown in Fig. 2 and the following module assignment. Operations $+_1$ and $+_2$ are assigned to module M_1 and operations $*_1$ and $*_2$ are assigned to module M_2. Thus $V_1 = \{ +_1, +_2 \}$ where each element is an instance of M_1 and $TM(M_1) = 2$.

Definition 3: The **input variable set** of module M_i, I_{M_i} is the set of all the operand variables associated with each instance j of module M_i. The **output variable set** of module M_i, O_{M_i} is the set of all the output variables associated with each instance j of module M_i.

For the scheduled DFG of Fig. 2 and the above mentioned module assignment $I_{M_1} = \{ a, b, c, d \}$ and $O_{M_1} = \{ d, f \}$.

A register assignment Π_R can be defined as a partition $\{ R_1, R_2, ..., R_r \}$ of the set of variables E such that for any

two variables u and v in R_k, $1 \leq k \leq r$, their lifetimes do not overlap. For the scheduled DFG of Fig. 2, a minimum of three registers are required. There are 108 distinct assignments of the variables in E to three registers. With respect to register and functional unit area these 108 assignments are equivalent. Only a subset of these result in more testable data paths (low BIST overhead) than the rest. Our algorithms direct register assignment to this low BIST area overhead subset of the solution space.

A. Maximizing sharing of test resources

Observation 1: Given a module assignment,
(a) An assignment of variables to a register R_i such that $R_i \cap I_{M_j} \neq \phi$ and $R_i \cap I_{M_k} \neq \phi$, guarantees the creation of *simple I-paths* to an input port of module M_j and to an input port of module M_k that share a common *head*, namely register R_i, *independent* of the subsequent interconnect assignment.
(b) An assignment of variables to a register R_i such that $R_i \cap O_{M_j} \neq \phi$ and $R_i \cap O_{M_k} \neq \phi$, guarantees the creation of *simple I-paths* from the output port of module M_j and from the output port of module M_k that share a common *tail*, namely register R_i, *independent* of the subsequent interconnect assignment.

Fig. 3 shows the formation of the simple I-paths with a common head and with a common tail. Consider a portion of a scheduled DFG shown in Fig. 3(a). Each of the operations are scheduled in a different control step and all the variables depicted have disjoint lifetimes. Let op_3 be one of the operations assigned to module M_1 and operations op_1 and op_2 be the only operators assigned to module M_2. Now $I_{M_2} = \{ a, b, p, q \}$ and $I_{M_1} = \{ x, y, \ldots \}$. I_{M_1} could have elements in addition to x and y depending on which other operations are assigned to M_1. Suppose each of the input variables of M_2 is assigned to a separate register, e.g., a to R_2, b to R_3, p to R_4 and q to R_5. Now if a variable from I_{M_1}, say x, is assigned to one of these registers, say R_2, a simple I-path is created from R_2 to an input port of M_1 also. This is shown in Fig. 3(b). The only requirement for M_1 and M_2 to have I-paths with a common head is that *at least* one variable each from I_{M_1} and I_{M_2} be assigned to the same register, i.e. there should be a register R_i such that $R_i \cap I_{M_1} \neq \phi$ and $R_i \cap I_{M_2} \neq \phi$. Similarly if *at least* one variable each from O_{M_1} and O_{M_2} is assigned to the same register, R_6 in this case, I-paths from the outputs of both the modules to R_6 are created.

In Fig. 3(b), the register R_2 can be used as a TPG for both the modules and the register R_6 can be used as a SA for both the modules by activating the appropriate I-paths. Thus for maximizing the sharing of TPGs between the input ports of modules an assignment Π_R is desirable such that for each R_i the number of input variable sets with which it has at least one variable in common is maximized. Similarly the number of output variable sets with which each R_i has at least one common variable should be maximized.

Definition 4: The **sharing degree** $SD(v)$ of a variable v is the sum of the number of modules for which v is an input variable and the number of modules for which v is an output variable.
If $v \in I_{M_j}$, let $X_j^v = 1$, else $X_j^v = 0$; if $v \in O_{M_j}$, let $Y_j^v = 1$, else $Y_j^v = 0$. Then $SD(v) = \sum_{j=1}^{m} (X_j^v + Y_j^v)$, where m is the total number of modules assigned.

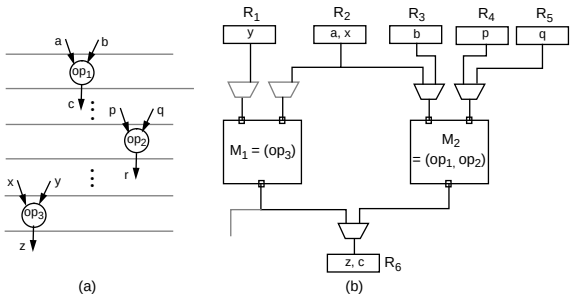


Fig. 3. Sharing of I-paths

Definition 5: The **sharing degree** of a register R is $SD(R) = \sum_{j=1}^{m} (X_j^R + Y_j^R)$, where

$$X_j^R = \bigvee_{\forall v \in R} X_j^v \quad and \quad Y_j^R = \bigvee_{\forall v \in R} Y_j^v$$

The sharing degree of a register is thus the total number of distinct input variable sets and distinct output variable sets that contain at least one element of R. The sharing degree reflects the number of modules for which the register can act as a TPG and the number of modules for which it can act as a SA. Consider a register R that has been assigned some variables. Let the sharing degree of R after another variable v is assigned to it be denoted by $SD(R, v)$. Now $SD(R, v) = SD(R) + SD(v) - \sum_{j=1}^{m} (X_j^R * X_j^v + Y_j^R * Y_j^v)$. Using this measure the assignment process can be guided by choosing merges that result in large increases in the sharing degrees of registers. The increase in the sharing degree of a register R as a result of assigning variable v to it will be denoted by $\Delta SD^v(R)$. (i.e. $\Delta SD^v(R) = SD(R, v) - SD(R)$).

The register assignment problem can be modeled as coloring of the *variable conflict graph*. A variable conflict graph has vertices corresponding to variables with an edge between two variables only if they have overlapping lifetimes. A coloring of this graph corresponds to a valid register assignment with each color corresponding to a register. In the rest of the paper we will use the terms color and register interchangeably. If the data flow graph description does not contain mutual exclusion constructs and loops, the resulting variable conflict graph is an interval graph [10]. A minimum coloring of these graphs can be obtained in polynomial time [11]. The minimum coloring algorithm on interval graphs is a greedy algorithm and at every step it has a restricted choice of variables which does not allow for an efficient exploration of the solution space to search for a good testability solution.

The greedy coloring algorithm uses the hereditary property of interval graphs which is defined through *simplicial* vertices. A vertex v of a graph G is simplicial if its *adjacency set* induces a clique in G. The adjacency set is the set of all vertices that are connected to v. An interval graph has at least two simplicial vertices. If a simplicial vertex and all its incident edges are removed, the remaining graph is also an interval graph. An ordering of the vertices such that each vertex is a simplicial vertex of the remaining graph is called a *perfect vertex elimination scheme* (*PVES*). An interval graph has many such perfect vertex elimination schemes. The optimal coloring algorithm constructs one such scheme *arbitrarily* and

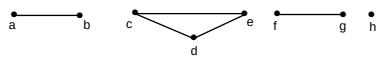
colors the vertices *greedily* in the reverse order (*reverse PVES*) [11]. Our heuristic is different from the optimal coloring algorithm in two respects: 1) it selects the $PVES$ in a more structured way taking into account information such as the sharing degree of variables and size of maximal cliques; and 2) the vertices are then colored using this scheme, but instead of assigning colors greedily, many more coloring possibilities are explored and the one most suited for maximizing the sharing of test resources for BIST is selected.

1. Selection of a $PVES$: With each vertex of the conflict graph we associate a sharing degree as per Definition 4. In addition we also find the size of the maximum clique to which each vertex belongs. The size of such a clique indicates the number of registers to which this variable *cannot* be assigned. Let $MCS(v)$ denote the size of such a clique containing v. The vertices are ordered such that if v is before w, then $SD(v) \leq SD(w)$ and if $SD(v) = SD(w)$ then $MCS(v) \leq MCS(w)$. At every step of constructing a $PVES$ there is a choice of simplicial vertices. The $PVES$ is now determined such that at each step a simplicial vertex that is earliest in this order is selected. Since vertices are colored in the *reverse PVES* order, this results in vertices with higher sharing degrees to be considered earlier when there is maximum flexibility in the assignment of colors. Also since vertices with a higher MCS value are considered earlier more colors are fixed in the earlier stages thus creating more coloring options to explore. This enables the heuristic to search the design space more efficiently for finding a coloring with low testability area overhead keeping the number of colors close to optimum.

2. Coloring in reverse $PVES$ order: For the purposes of the following discussion the vertices will be referred to by their number in the reverse $PVES$. Let the assignment after the kth vertex is colored be denoted as $\Pi_R^k = (R_1^k, R_2^k, ..., R_{c_k}^k)$ where $R_i^k \cap R_j^k = \phi$ if $i \neq j$ and $\bigcup_{i=1}^{c_k} R_i^k = \{1, 2, ..., k\}$. The total number of registers after the kth vertex is colored is c_k.

The vertex $(k+1)$ is assigned in the following way. If $(k+1)$ conflicts with all registers $R_1^k, R_2^k, ..., R_{c_k}^k$ then a new register $R_{c_{k+1}}^{k+1} = \{k+1\}$ is created. Otherwise, out of the registers that do not conflict with $(k+1)$ pick a register R_i^k such that $\Delta SD^{k+1}(R_i^k) = SD(R_i^k, k+1) - SD(R_i^k)$ is maximum. Such an R_i^k corresponds to a register that can best utilize $(k+1)$ to improve its sharing as a test resource. If there is more than one such register then the tie is broken by considering the sharing degree of the registers and the one which has the higher sharing degree is chosen. Further ties are broken by taking into consideration the effect of the assignment on interconnect cost. There are two cases in which a register other than R_i^k might be preferable for assignment.

Case 1: Consider the following example with 3 modules M_1, M_2 and M_3 with output variable sets $O_{M_1} = \{a, b\}$, $O_{M_2} = \{c, d\}$ and $O_{M_3} = \{e, f\}$. Assume that the assignment so far is $R_1 = \{a, b, c\}$ and $R_2 = \{d, e\}$ and f is the vertex under consideration for coloring next. Furthermore let us assume that f does not conflict with any of the other variables so it can be assigned to either R_1 or R_2 and assume $\Delta SD^f(R_1) > \Delta SD^f(R_2)$. This implies that R_1 can make the best use of variable f in terms of increasing its potential as a test resource. The increase in its sharing potential is due to the fact that since $f \in O_{M_3}$,



Variables	v	:	a	b	c	d	e	f	g	h
Sharing degree	$SD(v)$	:	1	1	2	2	1	2	2	1
Max. clique size	$MCS(v)$	:	2	2	3	3	3	2	2	1

Fig. 4. Conflict graph of variables

R_1 can be a potential SA for testing module M_3. But register R_2 already has variable e assigned to it and $e \in O_{M_3}$. If $SD(R_2) \geq SD(R_1, f)$, there is a greater likelihood of R_2 being chosen as a test resource in which case it can act as a SA for module M_3. Assigning f to R_1 in such a case would only increase the interconnection cost since module M_3 would have to transfer data from its output to two registers instead of one.

Hence if vertex $(k+1)$ is an output variable of module M_j and if there is a register R_l^k with which $(k+1)$ does not conflict such that R_l^k already has an output variable of M_j assigned to it then the final sharing degrees of R_i^k and R_l^k are compared. If $SD(R_l^k) > SD(R_i^k, k+1)$ then $(k+1)$ is assigned to R_l^k instead of R_i^k even if $\Delta SD^f(R_i^k)$ is greater.

Case 2: This case is analogous to case 1 above except that it deals with the possibility of selection of registers as potential TPGs. Since operators are binary, this case requires the existence of *two* such registers, R_m^k and R_n^k. In general both the cases can arise in which case a set of candidate registers is created of all such registers R_l^k, R_m^k and R_n^k and $(k+1)$ is assigned to the one which results in the highest increase in its sharing degree. Again ties are broken so as to minimize interconnect.

The optimality in the minimum coloring algorithm is guaranteed by assigning the vertex $(k+1)$ to the *first* R_i^k with which it does not conflict. Since our heuristic does not make such an assignment we cannot guarantee optimality in terms of the number of registers allocated. However the heuristic is near-optimal since it still relies on a $PVES$ of a conflict graph. In all the examples considered it resulted in the minimum number of registers. The details of the register assignment algorithm and the pseudo-code can be found in [12].

Consider the scheduled DFG in Fig. 2 with the following module assignment: $M_1 = \{+_1, +_2\}$ and $M_2 = \{*_1, *_2\}$. The variable conflict graph along with the SD and MCS values is shown in Fig. 4. A perfect vertex elimination scheme of this graph satisfying the SD and MCS ordering is h, b, a, e, g, f, d, c. The coloring is now done in the reverse order, namely, c, d, f, g, e, a, b, h. The first two vertices c and d are assigned to separate registers since they conflict and thus we have $R_1^2 = \{c\}$, $R_2^2 = \{d\}$ and $SD(R_1^2) = 2$, $SD(R_2^2) = 2$. Consider the assignment of the third vertex, f. Since $\Delta SD^f(R_1^2) = SD(R_1^2, f) - SD(R_1^2) = 4 - 2 = 2$ which is greater than $\Delta SD^f(R_2^2) = SD(R_2^2, f) - SD(R_2^2) = 3 - 2 = 1$, f is assigned to R_1^2 and we have $\Pi_R^3 = (\{c, f\}, \{d\})$. Vertex g is assigned the same color as d. Vertex e conflicts with both the allocated registers and hence a new register is allocated and e is assigned to it. The sixth vertex, a, belongs to $I_{M_1} = \{a, b, c, d\}$. There are two registers $R_1^5 = \{c, f\}$ and $R_2^5 = \{d, g\}$ that have elements from I_{M_1} and sharing degrees 3 and 4, respectively. The *increase* in the

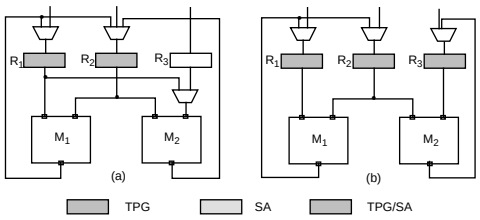


Fig. 5. Data paths from DFG in Fig.2.

sharing degree of R_3^5 due to a, $\Delta SD^a(R_3^5)$ is greater than $\Delta SD^a(R_1^5)$ and $\Delta SD^a(R_2^5)$. But the *actual* sharing degree $SD(R_3^5, a)$ is less than $SD(R_1^5)$ and $SD(R_2^5)$. Hence we prefer to assign a to R_1^5 or R_2^5. R_1^5 is chosen because of lower interconnect cost. The last vertex h increases the sharing degree of R_1^7 and R_3^7 by 1. However R_2^7 has an element of O_{M_2} to which h also belongs and $SD(R_2^7)$ is greater than $SD(R_1^7, h)$ and $SD(R_3^7, h)$. Hence h is assigned to R_2^7. The final assignment is $\Pi_R^8 = (\{c, f, a\}, \{d, g, b, h\}, \{e\})$.

The data path corresponding to this register assignment and the given module assignment is shown in Fig. 5(a). The data path corresponding to a minimum coloring obtained without regard for testability is shown in Fig. 5(b). In both cases minimum interconnect was assigned after register assignment.

It can be seen that in Fig. 5(a) R_1 and R_2 can be shared as TPGs between both the modules and R_2 can be shared as a SA. So a minimal area BIST solution would be to convert R_1 to a TPG and R_2 to a CBILBO. Compare this to Fig. 5(b) where R_2 is the only common TPG. Also the two modules do not have a SA in common. A minimal area BIST solution for this data path is R_1 as a TPG and R_2 and R_3 as CBILBOs which is more costly than the minimal area BIST solution for the earlier design.

B. *Minimizing CBILBOs*

Sharing test resources between modules is not sufficient to minimize the BIST area overhead because it could still result in the formation of data paths that require CBILBOs to test some modules. In a globally minimal BIST area overhead solution, a register might be modified into a CBILBO register even though it is not necessary to do so. However a situation where modifying a register to a CBILBO is *absolutely necessary* is the one which results in high BIST area overhead. In this section we derive the *exact* conditions for register assignment which when followed by *minimum* interconnect assignment necessitate the modification of a register to a CBILBO.

Definition 6: An **input register** of module M_k is a register R_i such that at least one input variable (operand) of M_k is assigned to it, i.e. $R_i \cap I_{M_k} \neq \phi$. An **output register** of module M_k is a register R_i such that at least one output variable of M_k is assigned to it, i.e. $R_i \cap O_{M_k} \neq \phi$. The set of input registers and output registers will be denoted by IR_k and OR_k respectively.

If a BIST embedding for a module is chosen such that the I-path from the output port of the module and the I-path to an input port of the module have the same register, a CBILBO has to be used. In general any register that is an input register as well as an output register of a module can be made a CBILBO in order to test the module. But it is *essential* for a register to be made a CBILBO *only if* it is a CBILBO in *all the possible* embeddings of a module. The conditions derived below are for a register to be a CBILBO in all possible BIST embeddings of the final data path.

Lemma 1: If all the possible BIST embeddings of module M_k require a CBILBO register then $| OR_k | \leq 2$.

From the above lemma and the definition of an output register, if register R_i has to be a CBILBO in any BIST embedding of module M_k then either (i) $R_i \cap O_{M_k} = O_{M_k}$, or (ii) $R_i \cap O_{M_k} \subset O_{M_k}$ and there exists a register R_j such that $(R_i \cap O_{M_k}) \cup (R_j \cap O_{M_k}) = O_{M_k}$. That is, the variables of the output variable set of M_k are assigned to either one or two registers. This information reduces the number of cases to be considered for deriving the exact assignment conditions for a register to be made a CBILBO.

Lemma 2: A register R_x is a CBILBO in all embeddings of module M_k if and only if one of the following cases is true.

Case(i): $R_x \cap O_{M_k} = O_{M_k}$ **and** $R_x \cap I^j_{M_k} \neq \phi$ for $j = 1, 2, ..., TM(M_k)$.

Case(ii): $R_x \cap O_{M_k} \subset O_{M_k}$ **and** $R_x \cap I^j_{M_k} \neq \phi$ for $j = 1, 2, ..., TM(M_k)$ and $\exists$ a register R_y such that $(R_x \cap O_{M_k}) \cup (R_y \cap O_{M_k}) = O_{M_k}$ **and** $R_y \cap I^j_{M_k} \neq \phi$ for $j = 1, 2, ..., TM(M_k)$. Case(ii) is symmetrical in R_x and R_y and so either of them can be made CBILBO.

The proofs of Lemma 1 and Lemma 2 are provided in [12]. Lemma 2 enables us to check if a particular assignment would result in a CBILBO in the BIST version of the design. The register assignment algorithm is modified to include the check and to avoid assignments leading to CBILBOs. If such an assignment cannot be avoided without allocating an extra register, we allow the assignment. Our experiments indicated that the assignment space is large enough for this situation to occur infrequently.

IV. Interconnect assignment

Register assignment is followed by minimum interconnect assignment. For a given module assignment different register assignments have different effects on interconnect area. Our register assignment algorithm does not take into account the effect on interconnect area except to resolve ties. The typical situations that occur when two variables or intermediate registers are merged into one register are shown in Fig. 6. The corresponding increase or decrease in multiplexers and BIST resources as a result of the merges is also shown.

Case 1: Merging variables/intermediate registers that have different source modules and different destination modules.(Fig. 6(a))

Case 2:Merging variables/intermediate registers where a source module of one variable is the destination module of the other variable.(Fig. 6(b))

Case 3:Merging variables/intermediate registers having only one destination module in common but different source modules.(Fig. 6(c))

Case 4:Merging variables/intermediate registers having only one source module in common but different destination modules.(Fig. 6(d))

TABLE I Design comparisons with BIST area overhead

DFG	Module Assignment	Traditional HLS			Testable HLS			% Reduction in BIST area
		# Reg	# Mux	% BIST area	# Reg	# Mux	% BIST area	
ex1	1+, 1*	3	3	18.14	3	3	10.67	30.00
ex2	1/, 2*, 2+, 1&	5	5	11.17	5	4	7.56	32.31
Tseng1	2+, 1*, 1-, 1&, 1|, 1/	5	9	17.65	5	7	11.34	35.75
Tseng2	1+, 3 ALUs	5	7	10.04	5	10	5.66	46.62
Paulin	1+, 2*, 1-	4	6	16.34	4	6	9.34	42.84

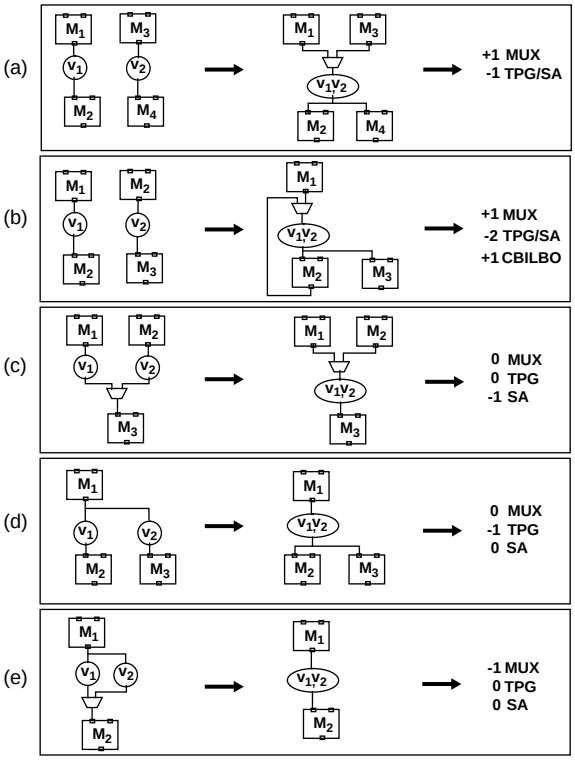


Fig. 6. Effect of register assignment on interconnect

Case 5:Merging variables/intermediate registers having both a common source module and a common destination module.(Fig. 6(e))

The above discussion shows that a register assignment with consideration for BIST overhead and without any consideration for interconnect area will still result in a data path with lower *overall* area. Our results indicated that in all cases the increase in the number of multiplexers, if any, was compensated for by the savings achieved by a reduction in the number of BIST resources. For example in Table I, the number of multiplexers has increased only in the case of Tseng2 which also has a high reduction in BIST area that of 46.62%.

For a particular register assignment, the minimum interconnect assignment solution space can be explored for further reduction in the BIST area overhead. Given a module M_k and the set of input registers IR_k, each input register can be connected in one of the following three possible ways: 1) it is only connected to the "left" input port of M_k, 2) it is only connected to the "right" input port of M_k, and 3) it is connected to both the "left" and the "right" input port of M_k. Assignment of interconnect Π_I can be thus seen as a partition of IR_k into sets IR_k^L, IR_k^R and IR_k^{LR} corresponding to the cases 1), 2) and 3), respectively. Pangrle has shown that the minimum connectivity assignment is one that minimizes $\mid IR_k^{LR} \mid$ [13]. For making the most use of the register assignment in reducing the BIST overhead, the connectivity assignment can be directed to ensure that registers with high sharing degrees have a better chance of being selected as TPGs. To test a module M_k, two registers with independent I-paths to distinct input ports of M_k have to be made TPGs. A register $R_i \in IR_k^{LR}$ has a better chance of being chosen as a TPG since it can serve as a TPG for either the left or the right port. Hence it is advantageous to have a register with a high sharing degree in IR_k^{LR}. The output connectivity assignment has no effect on the selection of BIST resources.

The connectivity assignment can be modeled as a double clique partitioning of the input register compatibility graph [13]. In this graph each input register is a vertex with an edge between two vertices if they can be connected to the same input port. The two disjoint cliques correspond to cases 1) and 2) and the rest of the vertices to case 3). We use weights in the clique partitioning algorithm to direct the assignment towards finding cliques such that registers with high sharing degrees are connected to both the input ports [12].

V. Results

Data paths from some scheduled DFGs and module assignments were synthesized using traditional assignment algorithms which optimize functional area and using the assignment algorithms presented in this paper. ex1 is the DFG from Fig. 2. ex2 is a DFG taken from [6]. Tseng1 and Tseng2 are different module assignments of the Tseng high-level synthesis benchmark [14]. Paulin is another standard high-level synthesis benchmark - the differential equation solver [15].

The data paths synthesized by the two approaches were then made testable using the Built-In Test System (BITS) of the USC-Test system [16]. BITS generates a variety of BIST designs for a data path depending upon which parameter is to be optimized. BITS was used to generate the minimal area BIST solutions for all the data paths. The BIST area overhead of these designs was then compared. The area overhead is in terms of gate count. Table I shows that our assignment resulted in 30-45% reduction in the BIST area overhead as compared to the traditional assignment. Note that the module assignment and the number of registers in both the cases are the same. In all the cases the number of registers is the minimum required. The BIST area overhead is expressed as a percentage increase in the gate count as a result of using the BIST registers from our library. Table II depicts the ac-

TABLE II Minimal area BIST solutions

DFG	Traditional HLS	Testable HLS
ex1	2 CBILBO, 1 TPG	1 CBILBO, 1 TPG
ex2	2 CBILBO, 1 TPG/SA, 2 TPG	1 CBILBO, 2 TPG/SA, 1 TPG
Tseng1	2 CBILBO, 3 TPG/SA	1 CBILBO, 3 TPG/SA, 1 TPG
Tseng2	2 CBILBO, 1 TPG/SA, 1 TPG	2 TPG/SA, 1 TPG
Paulin	3 CBILBO, 1 TPG/SA	1 CBILBO, 2 TPG, 1 SA

TABLE III Design comparison for Paulin example

HLS System	Module allocation	# Reg	# TPG	# SA	# BILBO	# CBILBO
RALLOC	1+, 2*, 1-	5	0	0	4	1
SYNTEST	(+*), (> *−), (*+)	5	4	1	0	0
Ours	1+, 2*, 1-	4	2	1	0	1

tual number of BIST resources and their modes used to make the data paths testable. It can be observed that using the assignment algorithms presented in this paper has significantly reduced the total number of BIST registers as well as the number of CBILBOs required to test the combinational modules in the data path.

We also compared our approach with two other synthesis for testability approaches, RALLOC [5] and SYNTEST [7]. Table III shows a comparison of BIST versions of data paths synthesized from the Paulin differential equation benchmark. The total number of registers allocated and the number of BIST registers required to make the data paths testable are shown in the table. Since the module allocations are different it is not possible to show the BIST area overhead as a percentage of the total area. But it can be clearly seen that our approach resulted in a smaller number of total registers as well as a smaller number of BIST registers.

VI. Conclusions

One of the considerations in applying BIST techniques to digital circuits is the extra area overhead incurred by modifications to registers. In this paper we have presented a high-level synthesis approach to make the BIST approach cost-effective. The proposed data path allocation algorithms synthesize circuits in which the sharing of BIST registers between functional modules is maximized and the number of CBILBOs required to test the data path is minimized. Experimental results on benchmark examples demonstrate the ability of our algorithms to generate low BIST overhead designs.

References

[1] P.R. Chalasani, S. Bhawmik, A. Acharya, and P. Palchaudhari. Design of Testable VLSI Circuits with Minimum Area Overhead. *IEEE Trans. on Computers*, pages 1460–1462, 1989.

[2] T. Lee, N. Jha, and W. Wolf. Behavioral Synthesis of Highly Testable Datapaths under the Non-scan and Partial Scan Environments. In *Proc. 30th Design Automation Conf.*, pages 292–297, June 1993.

[3] S. Dey, M. Potkonjak, and R.K. Roy. Synthesizing Designs with Low-Cardinality Minimum Feedback Vertex Sets for Partial Scan Application. In *Proc. VLSI Test Symp.*, pages 2–7, April 1994.

[4] A. Mujumdar, K. Saluja, and R. Jain. Incorporating Testability Considerations in High-level Synthesis. In *Proc. FTCS*, pages 272–279, 1992.

[5] L. Avra. Allocation and Assignment in High-level Synthesis for Self-testable Data Paths. In *Intn'l. Symp. on Circuits and Systems*, pages 463–472, Aug. 1991.

[6] C. Papachristou, S. Chiu, and H. Harmanani. A Data Path Synthesis Method for Self-Testable Designs. In *Proc. 28th Design Automation Conf.*, pages 378–384, June 1991.

[7] H. Harmanani and C. Papachristou. An Improved Method for RTL Synthesis with Testability Tradeoffs. In *Proc. Intn'l Conf. on Computer-Aided Design*, pages 30–35, Novemeber 1993.

[8] M.S. Abadir and M.A. Breuer. A Knowledge-Based System for Designing Testable VLSI Chips. *IEEE Design & Test of Computers*, pages 56–68, August 1985.

[9] L.T. Wang and E.J. McCluskey. Concurrent Built-In Logic Block Observer (CBILBO). In *Intn'l. Symp. on Circuits and Systems*, pages 1054–1057, 1986.

[10] D.L. Springer and D.E. Thomas. Exploiting the Special Structure of Conflict and Compatibility Graphs in High-Level Synthesis. In *Proc. Intn'l Conf. on Computer-Aided Design*, pages 254–257, Nov. 1990.

[11] M.C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, 1980.

[12] I. Parulkar. *Data Path Allocation Techniques for High-level Synthesis of Low BIST Area Overhead Designs*. CEng Tech. Report 95-02, Univ. of Southern California., Dept. of Elect. Engineering - Systems, April 1995.

[13] B.M. Pangrle. On the Complexity of Connectivity Binding. *IEEE Trans. on Computer-Aided Design*, pages 1460–1465, 1991.

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

[15] P.G. Paulin and J.P. Knight. Force-Directed Scheduling for the Behavioral Synthesis of ASICs. *IEEE Trans. on Computer-Aided Design*, pages 661–679, June 1989.

[16] S.P. Lin. *A Design System to Support Built-in Self Test of VLSI Circuits Using BILBO Oriented Test Methodologies*. Ph.D. thesis, Univ. of Southern California., Dept. of Electrical Engineering - Systems, May 1994.