Data Path Allocation Techniques for High-level Synthesis of Low BIST Area Overhead Designs

Ishwar Parulkar, Sandeep Gupta
and Melvin Breuer

CENG 95-02

Department of Electrical Engineering - Systems
University of Southern
Los Angeles, California 90089-2562
(213) 740-4469

April 1995

Data Path Allocation Techniques for High-level Synthesis of Low BIST Area Overhead Designs *

Ishwar Parulkar, Sandeep K. Gupta and Melvin A. Breuer

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 BIST registers. This paper proposes a high-level synthesis methodology that addresses this concern at an early stage in the design cycle. Data path allocation algorithms are presented that 1) maximize the sharing of registers as BIST resources resulting in a small number of registers being modified for BIST, and 2) minimize the number of CBILBO registers required in the BIST version. The designs synthesized by our algorithms have the same number of functional modules and registers as those synthesized using traditional approaches but with a much lower testability overhead.

*This work was supported by the Defense Advanced Research Projects Agency and monitored by the Federal Bureau of Investigation under Contract No. JFBI90092. The views and conclusions considered in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

1 Introduction

The increase in the complexity of VLSI chips has made the testing of these chips for physical faults an important as well as a difficult problem. With millions of transistors on a single chip and limited I/O the focus has shifted from external testing using automatic test pattern generation to built-in self-test (BIST) techniques. 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 to achieve a certain level of testing. How to reduce the BIST area overhead without sacrificing the quality of the test is an important research problem for test engineers [1], [2] and has been synthesized. 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 the 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. Lee et al. suggested a method of allocation and scheduling for "easy testability" that results in data paths with lower test generation time and higher fault coverage [3]. Dey et al. proposed an allocation method that resulted in partial scan designs that had a low area overhead [4]. An approach for synthesizing designs that have no self-loops and hence are cost-effective for partial scan was suggested by Mujumdar et al. [5]. For designs that support BIST, Avra proposed a register allocation method that minimizes the number of self-adjacent registers in the design [6]. The assumption in this work was 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 generated self-testable designs that did not have any self-loops [7]. The approach is based on constraining the allocation to generate a self-testable template and hence results in exploring a small subspace of the testable design space. Later they presented an improved method of generating self-testable designs by extending the self-testable template to include self-loops only in a specific configuration [8]. Though more general than the previous approach in terms of allowable templates, this approach is still restrictive because it merges ALUs, registers and interconnect simultaneously thus not utilizing the flexibility provided by the separate optimization of these sub-problems.

This paper describes a data path allocation scheme for generating designs that have a low BIST area overhead. Given a scheduled data flow graph (DFG) we allocate and assign 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 to make the designs testable is based on the concept of an I-path [9]. This model is more general and subsumes the testability design styles considered in design solution space is searched. Also the search is more efficient since operations,

variables and data transfers are assigned separately. The remainder of the paper is organized as follows. In Section 2 we present the testability model using the concept of I-paths. Sections 3 and 4 deal with the assignment of hardware to minimize test resources. Section 5 describes results that demonstrate the reduction in testability area overhead over designs synthesized without testability considerations.

2 Test methodology

The allocation scheme presented in this paper is directed towards synthesizing data paths that are to be tested using a pseudo-random BIST methodology. In this methodology, registers in the data path are reconfigured to support three modes of operation in addition to their normal mode: serial shift, test pattern generation (TPG) and parallel signature analysis (SA). The built-in logic-block observation (BILBO) register is a design capable of these modes [10]. During the test mode, a register configured as a TPG provides pseudo-random vectors to combinational logic, the response to which is compressed in a register configured as a SA. 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 [9]. The data path architectures comprising registers and combinational operator modules with busses and/or 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. The configuration of TPGs and SAs that can test a carry-lookahead adder can also test a ripple-carry adder or a multiplier. Hence the testability constraints for this BILBO BIST methodology are appropriate for consideration during high-level synthesis.

Definition 1 *(Abadir and Breuer [9]) 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. Thus the components of a simple I-path can only be multiplexers and a register (or a primary input/output).*

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

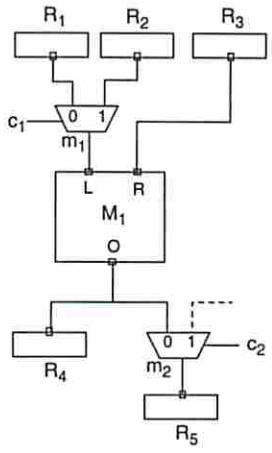


Figure 1: A generic configuration with simple I-paths

multiplexer m_1. I-paths with multiplexers can be activated by appropriate control signals. An I-path could have more than one multiplexer.

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 has to be used [11].

Generally, there is more than one embedding for a module. The choice of TPGS and SAs for testing this module largely depends on how these registers are connected with the rest of the modules. For example if there is a choice between registers for configuring as SA for a particular module, it is beneficial to chose the one that can also serve as a SA for some other module also or as a TPG for some other module. 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 allocation and assignment of variables to registers and the interconnect assignment 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.

3 Maximizing sharing of test resources

The behavioral description is assumed to be given in the form of a data flow graph (DFG) $G = (V, E)$ where 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 for operation $v \in V$, $S(v)$ corresponds to the control step in which 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. Various approaches to data path allocation with different ordering of these assignment subtasks have been studied [12]. Each has its merits and limitations. We chose the above order for the following reasons. The module assignment space is relatively much smaller than that for register and interconnect assignment. Also, the impact of module assignment on the functional area is large. Hence within the relatively small module assignment solution space there is not much flexibility for improving testability. Most of the flexibility for minimizing test resources for BIST exists in register assignment. Also, register assignments can be viewed as *potential* test resource assignments only *after* the module assignment is fixed. Interconnect assignment is strongly related to module and register assignment and its effect on functional area is 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. Let the total number of modules assigned be m.

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 *Let $I^j_{M_i}$ be the set of operand variables associated with instance j of module M_i. The* **input variable set** *of module M_i is $I_{M_i} = \bigcup I^j_{M_i}$, where the union is over all the instances j of*

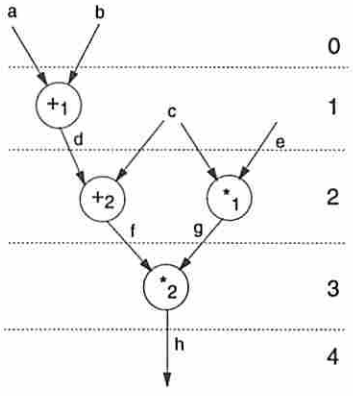


Figure 2: A scheduled DFG

M_i $(1 \leq j \leq TM(M_i))$. *Let* o_j *be the output variable associated with instance* j *of module* M_i. *The* **output variable set** *of module* M_i *is* $O_{M_i} = \bigcup \{ o_j \}$, *where the union is over all the instances* j *of* $M_i (1 \leq j \leq TM(M_i))$.

For the scheduled DFG of Fig. 2 and the same module assignment $I_{M_1} = I^1_{M_1} \cup I^2_{M_1} = \{a, b\} \cup \{d, c\} = \{a, b, c, d\}$. Similarly $O_{M_1} = \{d\} \cup \{f\} = \{d, f\}$.

3.1 Register assignment

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. We say that variable u is assigned to register R_k if $u \in R_k$. The minimum number of registers required is given by the maximum number of edges of a scheduled DFG cutting a control step boundary. In the scheduled DFG of Fig. 2, the boundary between control steps 1 and 2 is cut by three variables c, d and e and hence a minimum of three registers are required. For this minimum number of registers a large number of assignments are possible. There are $(3! \cdot 6 \cdot 6 \cdot 3) \div (3!) = 108$ distinct assignments of the variables in E to three registers. With respect to register and functional unit area all of these 108 assignments are equivalent. A subset of these are preferable in terms of interconnect complexity. Also 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. For this purpose each variable assignment is considered as a potential test resource assignment in addition to a functional storage value assignment. The assignment algorithm is based on the following two observations.

Observation 1 *Given a module assignment, if variables are assigned such that for some register* R_i, $R_i \cap I_{M_j} \neq \phi$, *then it can be guaranteed that a simple I-path from* R_i *to an input port of module* M_j *will be created, independent of the interconnect assignment.*

Similarly a variable assignment such that for some register R_i, $R_i \cap O_{M_j} \neq \phi$, *guarantees the creation of a simple I-path from the output port of module* M_j *to* R_i, *independent of the interconnect assignment.*

Since a variable from I_{M_j} is assigned to R_i the data transfer will be mapped on to either a simple interconnect wire from the register to an input port of the module or through a multiplexer. The second case will arise if another variable from I_{M_j} is assigned to a different register and the interconnect corresponding to this data transfer is bound to the same input port. In the first case R_i will definitely have to be modified as a TPG to test M_j while in the second case it might or might not be modified depending on the remaining variables assigned to R_i as well as the assignment of the remaining variables in I_{M_j}. Observation 2 follows as a direct consequence of Observation 1.

Observation 2 *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 interconnect assignment to follow.*
(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 interconnect assignment to follow.*

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 $I_{M_1} = \{ x, y, ... \}$. 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. Then from Observation 1 each of these registers have simple I-paths to input ports of M_2. 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). From Observation 1 this I-path exists irrespective of how the remaining variables in I_{M_1} are assigned. The only requirement for M_1 and M_2 to have I-paths with a common head is that *at least* one

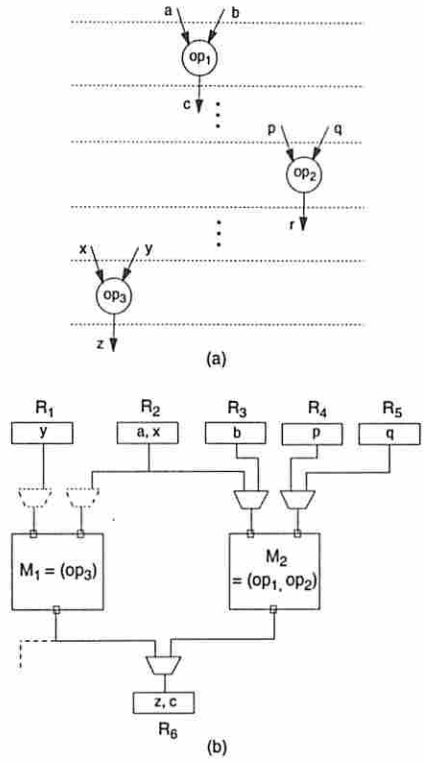


Figure 3: Sharing of I-paths

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 that it intersects is maximized. Similarly the number of output variable sets that intersect each R_i should be maximized.

We now define the *sharing degree* of a variable that will be used to determine a low BIST area overhead register assignment.

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.*

The sharing degree can be associated with a register also.

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

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

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 larger increases in the sharing degrees of registers over those resulting in lower increases thus requiring a fewer number of BIST resources. 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. Similarly the terms coloring and assignment are used interchangeably. Minimum coloring of general graphs has been proven to be NP-complete [13]. However polynomial time algorithms exist for special graphs such as chordal graphs and interval graphs [14]. If the data flow graph description does not contain mutual exclusion constructs and loops, the resulting variable conflict graph is an interval graph [15]. The polynomial time minimum coloring algorithm on interval graphs is a greedy algorithm. At every step the algorithm has a restricted choice of variables and does not allow for an efficient exploration of the solution space to search for a good testability solution. For a more thorough exploration we use a heuristic that is based on the greedy optimal algorithm but does not guarantee optimality in terms of number of registers. However our heuristic allocated the minimum number of registers in the examples tried.

The greedy coloring algorithm uses the hereditary property of interval graphs which is defined through *simplicial* vertices. A vertex v of $G = (V, E)$ is simplicial if its *adjacency set* $Adj(v)$ 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 with at least two simplicial vertices. An ordering of the vertices such that each vertex is a simplicial vertex of the remaining graph is called *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*) [16]. 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. However 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 in increasing order of their sharing degrees. Among vertices with the same sharing degree they are ordered in increasing order of the maximum clique sizes. Thus the ordering of the vertices is 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 the $PVES$ 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 coloring 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 colors after the kth vertex is colored is c_k Coloring vertex (k + 1) implies adding it to one of $R_1^k, R_2^k, ..., R_{c_k}^k$ or if it conflicts with all registers, creating a new set $R_{c_{k+1}}^{k+1} = \{\ k + 1\ \}$ to produce a new coloring Π_R^{k+1}. (Here, $c_{k+1} = c_k + 1$).

The vertex $(k+1)$ is colored 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(i): Suppose variable $(k+1)$ is an output variable of some module M_j. 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 and if $SD(R_l^k) > SD(R_i^k, k+1)$ then assigning $(k+1)$ to R_i^k does not help the situation. On the other hand it might increase the interconnection cost. So it is preferable to assign $(k+1)$ to R_l^k.

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 possibly assigned to either R_1 or R_2 and $\Delta SD^f(R_1) > \Delta SD^f(R_2)$. 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 $f \in O_{M_3}$, 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.

Case(ii): Suppose variable $(k+1)$ is an input variable of some module M_j. If among the non-conflicting registers there are two registers R_m^k and R_n^k such that each of them already has an input variable of M_j assigned then if their sharing degrees are higher than $SD(R_i^k, k+1)$, it is preferable to assign $(k+1)$ to one of them.

This case is analogous to case(i) except that it deals with the possibility of selection of registers as potential TPGs. Since we are assuming binary operators this case requires the existence of *two* such registers.

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 algorithm for the construction of $PVES$ is based on identifying simplicial vertices and maximal cliques to which they belong and sorting them according to their sharing degrees and maximal clique sizes. The register assignment algorithm *assign_reg(L)* is shown below. It takes as input a list of vertices L in the *reverse* $PVES$ order and outputs a register assignment Π_R. The algorithm iterates over each vertex v selected from L. Let $compat\_\Pi_R$ be the set of registers that are compatible (do not conflict) with variable v. Let $v \in I_{M_j}$ and/or $v \in O_{M_k}$. $input\_compat\_\Pi_R$ is a subset of $compat\_\Pi_R$ such that every register $R_i \in input\_compat\_\Pi_R$ is such that $R_i \cap I_{M_j} \neq \phi$. These are registers that have I-paths to the module for which v is an input variable. Similarly, $output\_compat\_\Pi_R$ is a subset of $compat\_\Pi_R$ such that every register $R_i \in output\_compat\_\Pi_R$ is such that $R_i \cap O_{M_k} \neq \phi$. These are registers that have I-paths from the module for which v is an output variable. Registers from $output\_compat\_\Pi_R$ and $input\_compat\_\Pi_R$ that satisfy cases(i) and (ii) above form the set of candidate registers *cand_reg*. The procedure *break_tie* selects a register from *cand_reg* with the maximum increase in sharing degree.

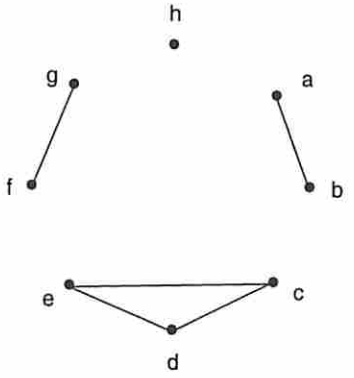
Consider the scheduled DFG in Fig. 2 with the following module assignment: $M_1 = \{ +_1, +_2 \}$ and $M_2 = \{ *_1, *_2 \}$. The sharing degree of the variables a, b, e and h is 1 and that of variables c, d, f and g is 2. The MCS value for variable h is 1, for variables a, b, f and g it is 2 and for variables c, d and e it is 3. The 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 $R_1^2 = \{ c \}$ and $R_2^2 = \{ d \}$. Consider the assignment of the third vertex, f. Since $SD(R_1^2, f) - SD(R_1^2) = 4 - 2 = 2$ which is greater than $SD(R_2^2, f) - SD(R_2^2) = 3 - 2 = 1$, f is assigned to R_1^2 and we have $\Pi_R^3 = \{ \{c, f\}, \{d\} \}$. Similarly the next vertex, g, is assigned the same color as d. The fifth 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 and R_2^5 that have elements from I_{M_1} and sharing degrees 3 and 4, respectively. The *increase* in the sharing degree of R_3^5 after assigning a to it is greater, but $SD(R_3^5, a) < SD(R_1^5)$ and $SD(R_3^5, a) < SD(R_2^5)$. 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 final coloring after all eight vertices have been colored is $\Pi_R^8 = \{ \{c, f, a\}, \{d, g, b, h\}, \{e\} \}$. 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.

Step 1: $\Pi_R = \Phi$
Step 2: **While** $(L \neq \Phi)$
    Step 2.1: $cand\_reg = \Phi$
    Step 2.2: Remove first vertex v from L
    Step 2.3: Find set of registers compatible with v, $compat\_\Pi_R$
    Step 2.4: **If** $(compat\_\Pi_R = \Phi)$
        Step 2.4.1: Allocate new register R_{new}
        Step 2.4.2: $R_{new} = \{ v \}$
        Step 2.4.3: $\Pi_R = \Pi_R \cup \{ R_{new} \}$
    Step 2.5: **Else**
        Step 2.5.1: Select $R_i \in compat\_\Pi_R$ such that $\Delta SD^v(R_i)$ is maximum
        Step 2.5.2: **If** $\exists R_l \in output\_compat\_\Pi_R$ such that $SD(R_l) > SD(R_i, v)$
            $cand\_reg = cand\_reg \cup \{ R_l \}$
        Step 2.5.3: **If** $\exists R_j, R_k \in input\_compat\_\Pi_R$ such that
            $SD(R_j) > SD(R_i, v)$ and $SD(R_k) > SD(R_i, v)$
            $cand\_reg = cand\_reg \cup \{ R_j, R_k \}$
        Step 2.5.4: **If** $(cand\_reg \neq \Phi)$
            $R \leftarrow break\_tie(cand\_reg)$
            $R = R \cup \{ v \}$
        Step 2.5.5: **Else**
            $R_i = R_i \cup \{ v \}$

Algorithm for register assignment - *assign_reg(L)*

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 the 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.



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

Figure 4: Conflict graph of variables

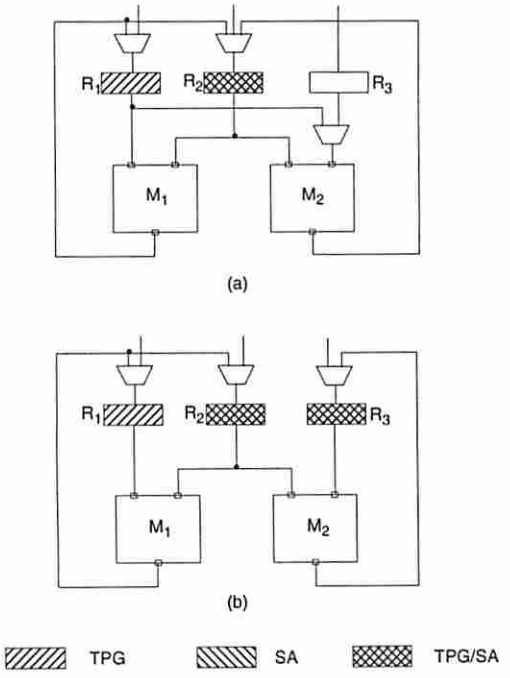


Figure 5: Two data path designs obtained with two register assignments

3.2 Interconnect assignment

The register assignment algorithm does not take into account the effect on interconnect area except to resolve ties. The area due to interconnect hardware is usually abstracted to the number of multiplexers. For a given module assignment different register assignments have different effects on interconnect area. 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.

1. Merging variables/intermediate registers that have different source modules and different destination modules results in the saving of one BILBO (TPG/SA) register but adds one multiplexer in front of the register. (Fig. 6(a))

2. Merging variables/intermediate registers where a source module of one variable is the destination module of the other variable results in an extra multiplexer. However BIST area is reduced since two BILBO registers are replaced by one CBILBO register. (Fig. 6(b))

3. (a) Merging variables/intermediate registers having only one destination module in common but different source modules results in one less SA without adding to interconnect complexity. (Fig. 6(c))

   (b) Merging variables/intermediate registers having only one source module in common but different destination modules results in one less TPG without adding to interconnect complexity. (Fig. 6(d))

4. Merging variables/intermediate registers having both a common source module and a common destination module has no effect on the BIST area overhead but it reduces the number of multiplexers by one. (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. However more reduction in the BIST area overhead can be achieved by assigning interconnect so as to make the most use of the register assignment. 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 $\mid IR_k^{LR} \mid$ [17]. For making the most use of the register

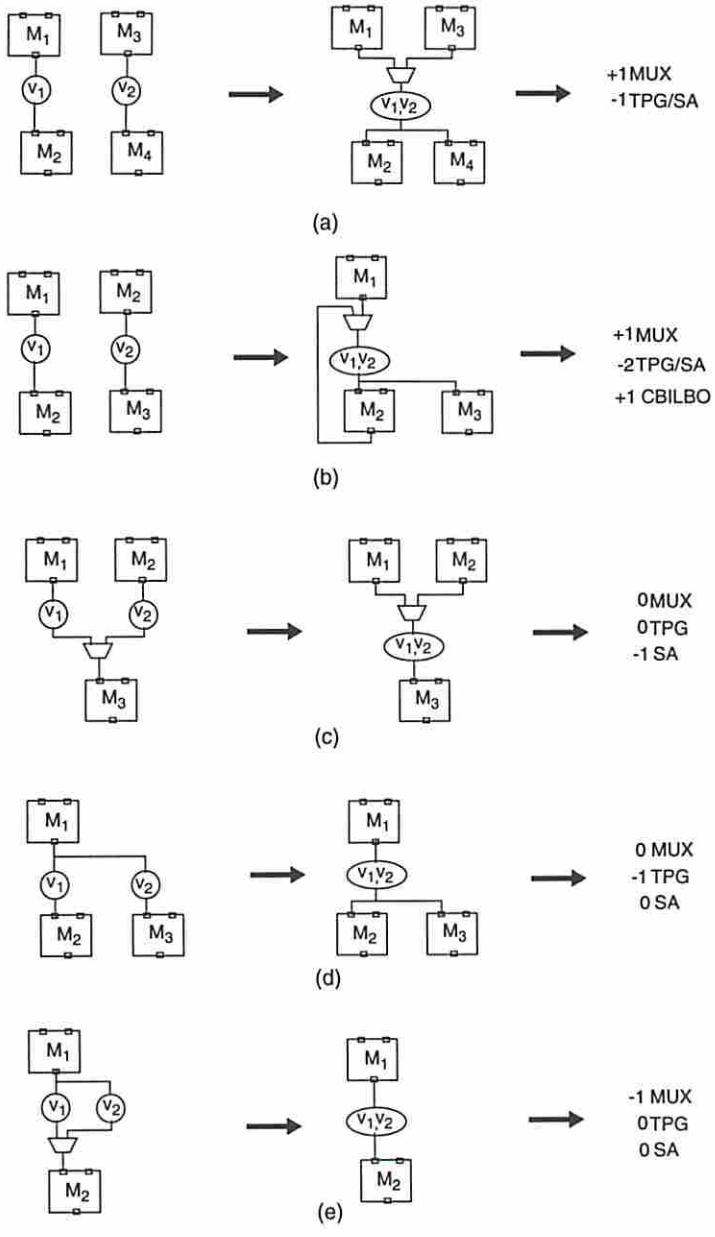


Figure 6: Effect of variable assignment on interconnect

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.

Consider module M_k with $I_{M_k}^1 = \{ a, b \}$, $I_{M_k}^2 = \{ c, d \}$, $I_{M_k}^3 = \{ e, f \}$ and $I_{M_k}^4 = \{ g, h \}$. Furthermore suppose that the register assignment is $R_1 = \{ a, e \}$, $R_2 = \{ b, c, g \}$, $R_3 = \{ h \}$ and $R_4 = \{ d, f \}$ and that the sharing degrees of these registers are 1, 1, 10 and 10, respectively. The minimum connectivity assignment for this configuration is one that has only one register connected to both the input ports of M_k. The connectivity assignment with R_2 connected to both the ports is shown in Fig. 7(a). R_1 or R_2 can be chosen as TPGs for the left port of M_k. But since the sharing degree of both these registers is 1, they won't be useful as TPGs or SAs for any other module. Now consider the minimum connectivity assignment shown in Fig. 7(b). R_2 or R_4 can be chosen as TPGs for the left port and R_4, R_1 or R_3 can be chosen as TPGs for the right port. R_4 has a very high sharing degree, 10, and since we have R_3 with a high sharing degree connected to the right port, we can select R_4 as a TPG for the left port and R_3 for the right port. In this configuration we were able to select both the TPGs with high sharing degrees because a register with a high sharing degree was connected to both the ports.

The connectivity assignment can be modeled as double clique partitioning of the input register compatibility graph [17]. 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 a weighted version of this algorithm to direct the assignment towards choosing registers with high sharing degrees to be connected to both the input ports.

4 Minimizing CBILBOs

The assignment algorithm for maximizing sharing of test resources does not have any control over the formation of data paths that have registers that need to be modified to CBILBOs to test some modules. A CBILBO register has an area approximately twice the area of a normal register and hence a data path with modules that do not require CBILBOs to test them would have a significantly lower BIST overhead. 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

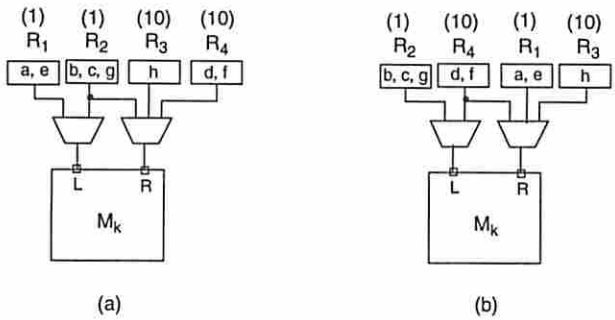


Figure 7: Effect of connectivity assignment on TPG selection

situation where modifying a register to a CBILBO is absolutely necessary is the one which results in high BIST area overhead. The algorithm presented in the previous section tries to maximize the sharing of test resources keeping the number of registers allocated close to minimum. A variant of the assignment algorithm that minimizes the number of CBILBOs while keeping the number of registers allocated the same is presented in this section.

We now derive conditions for register assignment which when followed by minimum interconnect assignment, necessitates the modification of a register to a CBILBO in the BIST version of the design.

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 the set of output registers of M_k will be denoted by IR_k and OR_k respectively.

The registers to be modified as TPGs for testing a module M_k are selected from IR_k and the registers to be modified as SAs from OR_k. A register is modified to a CBILBO only if the register is used as a TPG and SA for the same module. If a register R_i in the BIST version of a design is a CBILBO register then it is necessary that $R_i \in IR_k$ and $R_i \in OR_k$ for some k $(1 \leq k \leq m)$. If all the possible embeddings of a module use I-paths 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, then a CBILBO is essential to test the module. We call such a register an *essential CBILBO*. In general any register that is an input register as well as an output register of a module can be modified to a CBILBO in

order to test the module. But a register is an essential CBILBO only if it has to be modified into a CBILBO in all the possible embeddings of the module.

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

Proof: We will prove the contrapositive of the statement of the lemma - i.e. if $\mid OR_k \mid > 2$ then there exists at least one BIST embedding for M_k that does not require a CBILBO. Assume $\mid OR_k \mid > 2$. Select any $R_p, R_q \in IR_k$ to be TPGs. Since $\mid OR_k \mid > 2$, there exists at least one $R_r \in OR_k$ such that it is distinct from R_p and R_q. R_r can be selected as SA for M_k and we have an embedding without a CBILBO. □

From the above lemma and the definition of output register set, if register R_i has to made 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 CBILBO.

Lemma 2 *A register R_x is an essential CBILBO in order to test 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 proof of the lemma is provided in the Appendix for the reference of the reviewer. The above lemma enables us to check if a particular assignment would result in a CBILBO in the BIST version of the design. The check can be done in $O(\mid V \mid + max_k(TM(M_k)))$ time. The register assignment algorithm is modified to include the check and to avoid assignments leading to CBILBOs. The modified algorithm called *assign_reg_noCBILBO(L)* identifies the set of compatible registers $compat\_\Pi_R$ for a variable v just as in *assign_reg(L)*. However a check is performed to identify if assignment of v to a register from $compat\_\Pi_R$ creates the conditions defined in Lemma 2. Such registers are dropped from consideration for assignment and a subset $noCBILBO\_compat\_\Pi_R$ is created. The rest of the assignment procedure remains the same. It is possible that $noCBILBO\_compat\_\Pi_R = \Phi$ while $compat\_\Pi_R \neq \Phi$ which implies that a new register

Table 1: Area overhead of n-bit BIST registers

Type of register	Area overhead (# gates)
TPG	$7 + 5*(n-1)$
SA	$7 + 5*(n-1)$
BILBO	$9 + 7*(n-1)$
CBILBO	$14 + 12*(n-1)$

will have to be allocated. But since it is desirable to keep the total number of registers close to minimum, in a case like this, $noCBILBO\_compat\_\Pi_R$ is made the same as $compat\_\Pi_R$ implying that a CBILBO will be required. Our experiments indicated that the assignment space is large enough and this situation does not occur frequently.

5 Results

Data paths from some scheduled DFGs and module assignments were synthesized using the 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 [7]. Tseng1 and Tseng2 are different module assignments of the Tseng high-level synthesis benchmark [18]. Paulin is another standard high-level synthesis benchmark - the differential equation solver [19].

The data paths synthesized by the two approaches were then made testable using the Built-In Test System (BITS) of the USC-Test system [20]. 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. From the library used by BITS system, the area overhead incurred in converting a n-bit register in terms of the number of gates is given in Table 1.

Table 2 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 the library.

Table 3 depicts the actual number of BIST resources and their modes used to make the data

Table 2: Design comparisons with BIST area overhead

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

Table 3: 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

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.

Fig. 8 shows the BIST versions of data path synthesized from the Tseng2 data flow graph. Fig. 8(a) is synthesized using the traditional assignment methods while (b) is synthesized using our assignment heuristics.

We also compared our approcah with two other synthesis for testability approaches, RALLOC [6] and SYNTEST [8]. Table 4 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 was 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.

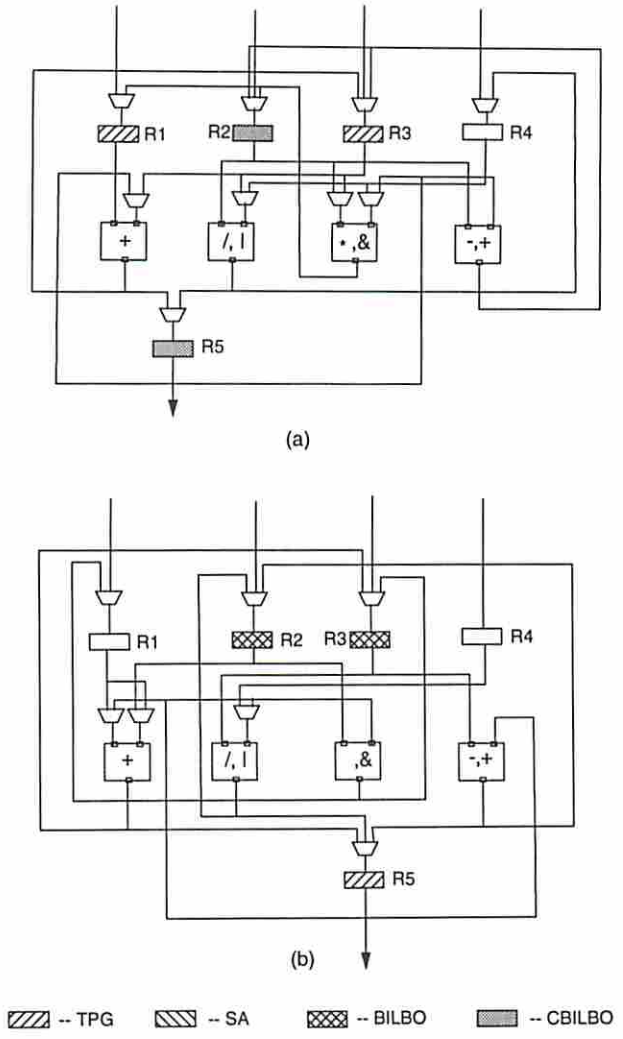


Figure 8: Comparison of BIST versions of Tseng2

Table 4: 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

6 Conclusions

Built-in self-test (BIST) techniques are becoming popular for testing digital circuits. BIST involves modification of registers in the design to perform test pattern generation, test response compaction, etc. One of the considerations in BIST techniques is extra area overhead incurred by these modifications. 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] G. Craig, C. Kime, and K. Saluja. Test Scheduling and Control for VLSI Built-in Self-test. *IEEE Trans. on Computers*, pages 1099–1109, 1988.

[3] 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.

[4] 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.

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] B. Koenemann, J. Mucha, and G. Zweihoff. Built-in Logic Block Observation Techniques. In *Proc. Intn'l. Test Conf.*, pages 37–41, Oct. 1979.

[11] 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.

[12] P. Michel, U. Lauther, and P. Duzy. *The Synthesis Approach to Digital System Design.* Kluwer Academic Publishers, 1992.

[13] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, 1979.

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

[15] 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.

[16] F. Gavril. Algorithms for Minimum Coloring, Maximum Clique, Minimum Covering by Cliques, and Maximum Independant Set of a Chordal Graph. *SIAM J. Computing*, pages 180–187, June 1972.

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

[18] 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.

[19] 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.

[20] 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.