Estimating BIST Resources in High-level Synthesis

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

CENG Technical Report 96-06

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

March 1996

Estimating BIST Resources in High-level Synthesis *

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

Abstract

Estimation of resources at various stages of the high-level synthesis process is essential to guide high-level synthesis algorithms towards optimal solutions. Lower bound estimation bounds the design space and gives an indication of the quality of the design synthesized. Previous work in high-level synthesis focused on bounds on *functional* resources. In this paper, we present lower bounds on the number of *test* resources (i.e. test pattern generators, signature analyzers and CBILBO registers) required to test the synthesized data path by the partial intrusion built-in self-test (BIST) methodology. The estimation is performed on scheduled data flow graphs and provides a practical way of selecting or modifying module assignments and schedules such that the synthesized data path requires a small number of test resources to test it.

*This work was supported by the Advanced Research Projects Agency and monitored by the Department of the Army, Ft.Huachuca, under Contract No. DABT63-95-C-0042. The information reported here does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred.

1 Introduction

Estimation of data path resources during high-level synthesis is essential for two reasons. First, it enables the designer to evaluate the design quality by comparing the estimates of any design metric with the constraints specified for that metric. For example, if the estimated number of functional resources in the design corresponds to an area that is greater than the area constraint, then the schedule may have to be modified to allow a functional resource allocation that is within the area constraint. Second, estimates enable the designer to explore design alternatives by providing quick feedback for any design decision. This way a designer can explore a greater number of alternatives instead of synthesizing a complete implementation and measuring the particular design metric for each design alternative. Lower bounds on resources not only greatly reduce the size of the solution space but also provide a means to measure the proximity of the final solution to the optimal one.

There is some recent work for estimating lower bounds on functional resources [1],[2],[3],[4],[5], [6]. These works are concerned with *functional* resources such as registers, adders and multipliers. For making a synthesized design testable using a built-in self-test (BIST) strategy, some of the registers in the design have to be modified into test pattern generators (TPGs), signature analyzers (SAs), built-in logic block observers (BILBOs) or concurrent BILBOs (CBILBOs). One of the considerations in BIST techniques is the extra area needed for these test resources. How to reduce the BIST area overhead without sacrificing the quality of the test is an important research problem [7]. A number of high-level synthesis approaches that incorporate BIST have been investigated in the recent past [8],[9],[10],[11],[12]. In these approaches allocation is done to reduce the BIST area overhead. Cost functions and heuristics are used to guide the allocation algorithms towards low BIST area overhead designs. None of the approaches have a mechanism for estimating the effect of a decision on the final number of test resources required. We believe this is the first work on estimating lower bounds on *test resources* for self-testable data paths in high-level synthesis.

In [12] an allocation approach was presented that was based on maximizing the sharing of registers as BIST resources and minimizing the number of CBILBO registers required in the BIST version of the synthesized design. This work assumed that a scheduled data flow graph was given and that module assignment was done without regard for testability overhead. However assuming a module assignment for the given scheduled DFG, limits the degree of sharing of test resources and CBILBO minimization that can be done by the approach suggested in [12]. In other words, for each selection of a module assignment for a scheduled DFG, there are lower bounds on the number of test resources that will be needed to test all the modules.

In this paper, we derive the lower bounds on test resources for BIST. The lower bounds are tight in the sense that they can be achieved if there are no functional resource constraints. The lower

bounds can be used as an estimate to determine the quality of the schedule and module assignment in terms of BIST area overhead of the synthesized design. The lower bound estimation technique can be used on partial schedules and module assignments to guide them towards testability optimal solutions. The bounds also serve as an independent measure of comparing the performance of different high-level synthesis systems and algorithms that perform testability optimization.

The remainder of the paper is organized as follows. In Section 2 we present the the testability model and some basic definitions. Section 3 deals with derivation of lower bounds on the number of signature analyzers, test pattern generators and CBILBOs. The computational complexity of the lower bounds and their use in high-level synthesis algorithms is discussed in Section 4. Section 5 describes data relating to lower bounds on the test resources of some high-level synthesis benchmarks.

2 Preliminaries

2.1 Test methodology

The high-level synthesis process assumed in this work is directed towards synthesizing data paths that are to be tested using a partial intrusion pseudo-random BIST methodology. A structural model of register-transfer level (RTL) designs synthesized by high-level synthesis systems is shown in Fig. 1. In partial intrusion BIST, a subset of registers are used in the test mode and all the functional modules and part of the steering logic and interconnections are tested as depicted in Fig. 1. In the test mode, some of the registers in the data path are reconfigured to support test pattern generation (TPG), and some to support signature analysis (SA). By appropriate selection of test resources (TPGs and SAs) all the functional modules in the design can be tested. In the process, some of the interconnections (multiplexer paths and wires) are also tested. The rest of the data path is assumed to be tested using functional tests. Note that in this partial-intrusion BIST methodology the test resources and paths used to generate, transport and collect test data are a subset of the functional data path. No additional data path components such as registers, multiplexers or interconnect are added for the purpose of testing.

For testing all the functional modules in the design using a small number of test resources, different mappings of registers to TPGs and SAs need to be considered. The concept of an I-path can be used effectively to explore the various mappings [13]. 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.

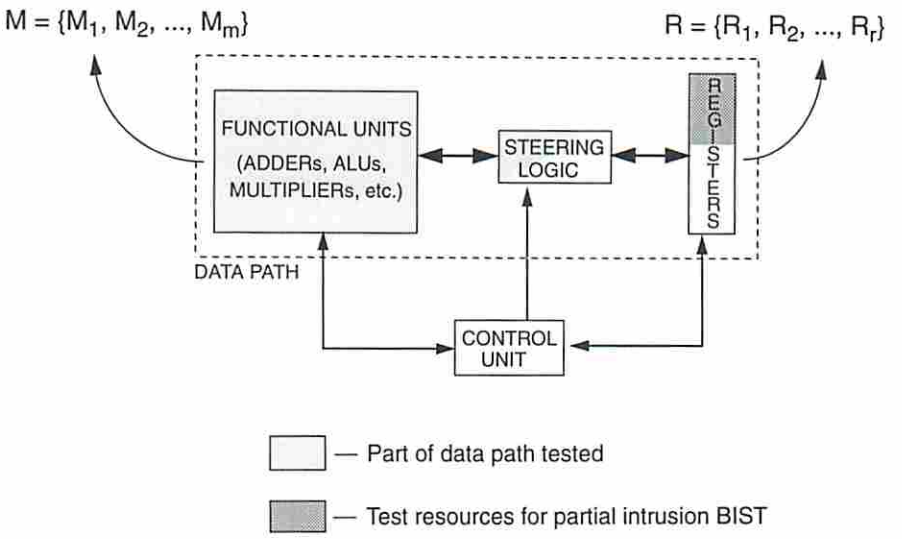

Figure 1: Test methodology

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. 2. Input port B has a simple I-path from R_3. This I-path is active at all times and is the only I-path to port B. Input port A 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. Embeddings for modules could be chosen such that a register that is a TPG for a module is an SA for a different module. In this case the register acts as a TPG and SA at different times and has to be modified to a BILBO register. The BILBO register has a higher overhead than a TPG or a SA. If the embedding chosen is such that a register is a TPG and an SA for the *same* module 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 [14].

Functional constraints determine the sharing of modules (portion of data path to tested) by operations and the sharing of registers (data path components to be used as test resources) by variables . The functional constraints thus impose lower bounds on the number of functional resources that can be used as test resources. If additional registers and/or interconnections (not used in the functional mode) were to be used to test a circuit, only two TPGs and one SA would

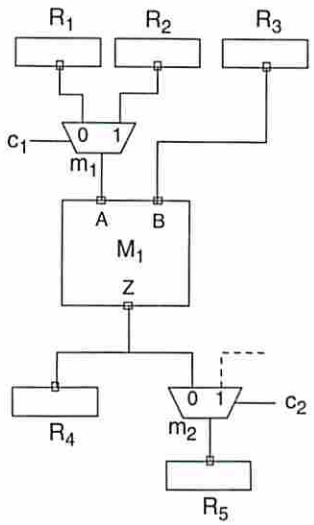

Figure 2: A generic configuration with simple I-paths

be required, assuming binary operations. But since test resources are selected from the set of functional registers this is not the case. Another point to note is that the objective of our testability optimization is BIST *area* overhead. Hence test concurrency (number of modules that can be tested simultaneously) does not depend on functional concurrency but rather on the sharing of test resources between modules.

2.2 Notation and definitions

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 $S(v)$ corresponds to the control step in which operation v is scheduled. Single operation per clock cycle is assumed and the synthesized data path is non-pipelined. All operators are assumed to be binary and commutative for the purpose of discussion in this paper. Non-commutative operators can be handled by adding additional constraints. Unary operators can be treated as a special case of binary operators. The module assignment is defined as $\Pi_M : V \rightarrow M$ 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. Π_M can be viewed as a partition $\{M_1, M_2, ..., M_m\}$ of the set of operations V into m modules.

Definition 1 *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. 3 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 $TM(M_1) = 2$.

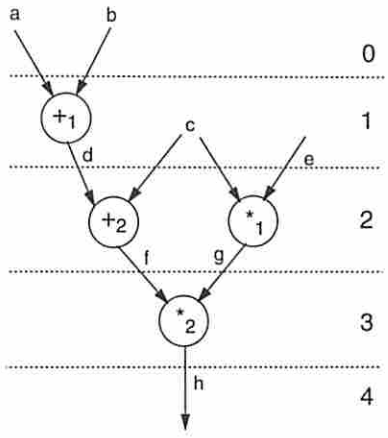

Figure 3: A scheduled DFG

Definition 2 *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. 3 and the above mentioned module assignment $I_{M_1} = \{a, b, c, d\}$ and $O_{M_1} = \{d, f\}$.

The following observation has been made in [12].

Observation 1 *Given a schedule and 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.*

For a module, any register to which an input variable is assigned can be used as a TPG and any register to which an output variable is assigned can be used as an SA. The distribution

of the elements of the input variable sets and output variable sets of modules determines the possible candidates for TPGs and SAs for modules. In determining a minimal area BIST solution for a design, different BIST embeddings for a module are explored. Embeddings that share test resources between modules are desired since they reduce the total number of test resources required to test the data path. Thus for maximizing the sharing of TPGs between the input ports of modules a register 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. Observation 1 identifies the conditions that can be used to perform such register assignments. Also a CBILBO register is expensive since its area is approximately twice that of a normal register. 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* for good fault coverage is the one which results in high BIST area overhead. A detailed description of the conditions for maximizing sharing of test resources and minimizing essential CBILBOs during register and interconnect assignment and the associated algorithms is given in [12],[15].

3 Lower bounds on test resources

A necessary condition for a module assignment to produce a valid circuit implementation is that the operations corresponding to a shared resource (i.e. all $v \in M_i$) do not execute concurrently. A schedule determines the module assignment space. Module assignment in turn determines the input and output variable sets of modules. The variables in these sets can be distributed across registers to ensure maximum sharing of registers as test resources between modules resulting in low BIST area overhead. A schedule also determines the lifetimes of variables and affects their assigning compatibility to registers. Thus a schedule and a module together assignment constrain the register and interconnect assignment in terms of the potential of sharing registers as test resources and avoiding essential CBILBOs.

The module assignment space is much smaller than the register and interconnect assignment space because each operation has an *operation type*. Only certain modules that can implement that operation type can be assigned to that operation. This is in contrast to register and interconnect assignment where there is only one type of hardware to be assigned - registers for storage values and wires for data transfer. Hence there is a great deal of flexibility in register and interconnect assignment in terms of optimizing for BIST area overhead. However, the optimum that can be achieved is bounded by the schedule and module assignment. Typically, several schedules and

module assignments that have a desirable latency and functional area, can differ significantly in their test resource requirement. For a given schedule and module assignment, establishing what can be achieved in terms of register and interconnect assignment to minimize test resources, is a key question in incorporating testability overhead optimization techniques in the scheduling phase of high-level synthesis.

3.1 Lower bounds on the number of TPGs and SAs

We address the following two questions. Given a scheduled DFG and a module assignment Π_M, among all register and interconnect assignments that can be associated with the given schedule and module assignment

1. what is the *lower bound* on the number of TPGs required to generate patterns to test all the modules, and
2. what is the *lower bound* on the number of SAs required to compress test responses for all the modules?

Note that we are interested in finding the lower bounds on the BIST area while relaxing the functional area constraints. The *total number* of registers or amount of interconnect required to achieve this bound is not being considered. A data path that achieves these bounds could have a higher functional area than another data path that might not meet the bounds on the number of TPGs or SAs.

Consider the following example of a scheduled DFG shown in Fig. 4(a) and the following two module assignments.

1. Assignment I:(Fig. 4(b)) Operations '$+_1$' and '$+_3$' are assigned to one module and operation '$+_2$' is assigned to a second module. $M_1 = \{+_1, +_3\}$ and $M_2 = \{+_2\}$.
2. Assignment II:(Fig. 4(c)) Operations '$+_1$' and '$+_2$' are assigned to one module and operation '$+_3$' is assigned to a second module. $M_1 = \{+_1, +_2\}$ and $M_2 = \{+_3\}$.

Consider all possible register and interconnect assignments for the above two module assignments. Any register which is assigned at least one variable from the output variable set of a module can be used as a SA for that module (from Observation 1 in Section 2).

For Assignment II, variables a and c can be assigned to the same register since their lifetimes do not overlap and this register can be used as a SA to test both M_1 and M_2 since $a \in O_{M_1}$ and $c \in O_{M_2}$. Hence the lower bound on the number of SAs in this case $LB_{\#SAs} = 1$.

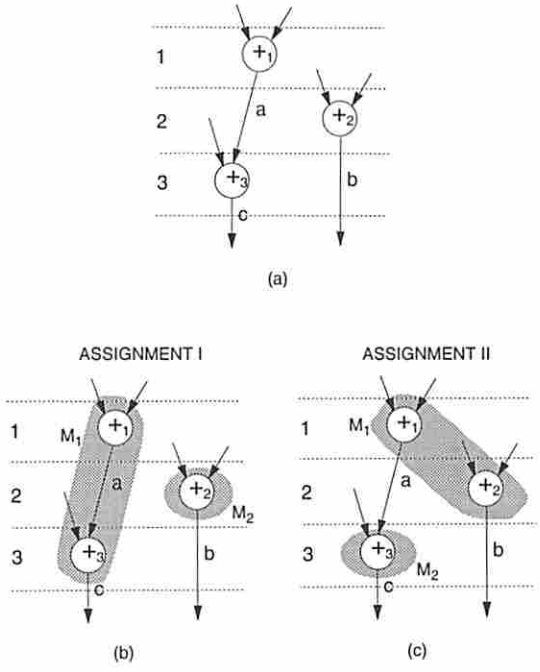

Figure 4: Effect of module assignment on $LB_{\#SAs}$ and $LB_{\#TPGs}$

For Assignment I it is not possible to find such a register assignment. In this case $O_{M_1} = \{a, c\}$ and $O_{M_2} = \{b\}$. Since the lifetime of b overlaps with the lifetimes of both a and c it cannot be assigned the same register to a register to which a or c is assigned. Hence for any register assignment the minimum number of SAs required to test M_1 and M_2, $LB_{\#SAs} = 2$.

An analogous situation occurs in the case of the input variables of operations corresponding to the lower bound on the number of TPGs. We have defined the concepts of *output storage concurrency*, *inputs storage concurrency* and *maximal concurrent operation set* in this paper that help model the dependence of BIST area on schedules and module assignments.

Definition 3 *a) The output variable of an operation $v \in V$ is the variable corresponding the outgoing edge of v in G. For a scheduled DFG, $G = (V, E)$ the* **output storage concurrency** *of a subset of operations $Q \subseteq V$, denoted by $C_{OS}(Q)$, is the maximum number of output variables of operations in Q alive at the same time.*

b) An input variable of an operation $v \in V$ is a variable corresponding to an incoming edge of v in G. For a scheduled DFG, $G = (V, E)$ the **inputs storage concurrency** *of a subset of operations $Q \subseteq V$, denoted by $C_{IS}(Q)$, is the maximum number of input variables of operations in Q alive at the same time.*

The output storage concurrency is a measure of the minimum number of storage locations that would be required to store the outputs of the operations in Q. For the scheduled DFG in Fig. 4 for the subset of operations $Q_1 = \{+_1,+_2\}$, the output storage concurrency $C_{OS}(Q_1) = 2$ since *both* the output variables a and b are alive at the same time. Similarly for $Q_2 = \{+_1,+_3\}$, $C_{OS}(Q_2) = 1$ and for $Q_3 = \{+_1,+_3,+_2\}$, $C_{OS}(Q_3) = 2$.

Definition 4 *Given a scheduled DFG, $G = (V, E)$ and a module assignment $\Pi_M = \{M_1,M_2,..., M_m\}$ a* **maximal concurrent operation set** *$V_{C_{max}}$ is a set of m operations, each of which is assigned to a different module.*

A maximal concurrent operation set refers to the maximal set of operations that can be executed simultaneously in real time because each of them have a dedicated hardware resource to which they are mapped by the module assignment. Note that in the actual behavior (scheduled DFG) these operations might not be executing concurrently. However the module assignment has assigned resources such that they *can* execute concurrently. For Assignment II discussed previously there are *two* maximal concurrent operation sets $V^1_{C_{max}} = \{+_1,+_3\}$ and $V^2_{C_{max}} = \{+_2,+_3\}$.

Using the concepts of *maximal concurrent operation set*, *output storage concurrency* and *inputs storage concurrency* we can compute the lower bound on the number of SAs and TPGs required for any data path synthesized from a particular schedule and module assignment.

Theorem 1 *For a given scheduled DFG, $G = (V, E)$ and a module assignment Π_M, the lower bound on the number of SAs required to test all the modules in the data path is given by*

$$\min_{\forall V_{C_{max}}} C_{OS}(V_{C_{max}})$$

where the minimum is over all maximal concurrent operation sets of Π_M.

Proof:

$$Let \quad \min_{\forall V_{C_{max}}} C_{OS}(V_{C_{max}}) = n$$

and the total number of modules assigned be m. Assume that there exists a way of assigning variables to registers such that the m modules can be tested using n' SAs and $n' < n$.

For a register to be a SA for a module M_i, at least one variable from output variable set O_{M_i} must be assigned to that register. Since m modules can be tested using n' SAs, there exist m variables each belonging to a distinct output variable set O_{M_i} $(i = 1, 2, ..., m)$ such that they can be assigned to n' registers. Let the set of operations that have a variable from these m variables as an output variable be denoted by V_c. Each of the operations in V_c is mapped to a distinct module

and $| V_c | = m$. Hence V_c is a maximal concurrent operation set from Definition 4. Since the output variables of operations in V_c can be assigned to n' registers, the output storage concurrency of V_c, $C_{OS}(V_c) \leq n'$. Since V_c is a maximal concurrent operation set of the given module assignment, this contradicts the fact that

$$\min_{\forall V_{C_{max}}} C_{OS}(V_{C_{max}}) = n.$$

Hence the lower bound on the number of SAs required to test the m modules is n. □

For Assignment I in the previous example, the maximal concurrent operation sets are $V^1_{C_{max}}$ = $\{+_1,+_2\}$ and $V^2_{C_{max}} = \{+_3,+_2\}$. Hence the lower bound on the number of SAs, $LB_{\#SAs}$ = min { $C_{OS}(V^1_{C_{max}}), C_{OS}(V^2_{C_{max}})$ } = min {2, 2} = 2. For Assignment II, the maximal concurrent operation sets are $V^1_{C_{max}} = \{+_1,+_3\}$ and $V^2_{C_{max}} = \{+_2,+_3\}$. Hence $LB_{\#SAs}$ = min { $C_{OS}(V^1_{C_{max}}), C_{OS}(V^2_{C_{max}})$ } = min {2, 1} = 1.

Similarly the lower bound on the number of TPGs can be found.

Theorem 2 *For a given scheduled DFG $G = (V, E)$ and a module assignment Π_M, the lower bound on the number of TPGs required to test all the modules in the data path is given by*

$$\min_{\forall V_{C_{max}}} C_{IS}(V_{C_{max}})$$

where the minimum is over all maximal concurrent operation sets of Π_M.

Proof: Similar to proof of Theorem 1.

The lower bounds derived above are *tight*. Given complete flexibility in assigning registers and interconnect without any area constraint, these bounds can be achieved. Theorems 1 and 2 indicate that to lower the BIST area overhead of the synthesized data path a module assignment that has a maximal concurrent operation set with low output (and inputs) storage concurrency is preferable.

3.2 Lower bounds on CBILBOs

A CBILBO is required to test a module if all possible BIST embeddings of the module use the same register as TPG and SA. It is a *necessary* condition for a register to be a self-adjacent register in order to be a CBILBO. Hence to determine the lower bound on the number of CBILBOs we first need to know the module assignment condition that when followed by *any* possible register and interconnect assignment results in a self-adjacent register.

Lemma 1 *Given a scheduled DFG, $G = (V, E)$ and a module assignment $\Pi_M = \{M_1, M_2, ..., M_m\}$ a self-adjacent register is created in the data path irrespective of the register and interconnect assignments if and only if there exists a module M_i such that $I_{M_i} \cap O_{M_i} \neq \phi$. The registers to which any element(s) of $I_{M_i} \cap O_{M_i}$ is assigned are self-adjacent registers.*

Proof:

(If):

Let there be a module M_i such that $I_{M_i} \cap O_{M_i} \neq \phi$. Let variable $x \in I_{M_i} \cap O_{M_i}$ and let R be the register to which x gets assigned. Since $x \in I_{M_i}$, therefore R is an input register of module M_i. Since $x \in O_{M_i}$, therefore R is also an output register of module M_i. From the definition of a self-adjacent register, R is a self-adjacent register.

(Only if):

Let there be a self-adjacent register R in the data path associated with module M_i. Since R is an input register as well as an output register of module M_i, therefore a variable from I_{M_i} (say x) and a variable from O_{M_i} (say y) are assigned to register R. But x and y have non-overlapping lifetimes and can always be assigned to different registers in an alternative register assignment precluding a self-adjacent register. The only possibility for x and y to be assigned to the *same* register in *all possible* register assignments is when $x \equiv y$ (i.e. x and y are the same variable). Hence $x \equiv y \in I_{M_i}$ and $x \equiv y \in O_{M_i}$, implying $I_{M_i} \cap O_{M_i} \neq \phi$. □

Self-adjacency is only a necessary condition for the CBILBO requirement. The temporal multiplicity of a module can be useful in avoiding CBILBOs in spite of the presence of self-adjacent registers [12]. We have defined the concept of an *essential concurrent operation* and *storage concurrency* to find lower bounds on the number of CBILBOs.

Definition 5 *An operation is an* **essential concurrent operation** *if it is an element of* all *maximal concurrent operation sets of a module assignment.*

From the definition of a maximal concurrent set, the module to which an essential concurrent operation is assigned has only *one* operation assigned to it, i.e., the temporal multiplicity of the module is 1. The condition for essential CBILBO is stated next as Theorem 3. To prove the condition we use the following lemma that has been proved in [12].

Lemma 2 *If all the possible BIST embeddings of module M_k require a CBILBO register then M_k has no more than 2 output registers.*

Theorem 3 *Given a schedule and a module assignment* $\Pi_M = \{M_1, M_2, ..., M_m\}$ *a CBILBO is essential to test module* M_i *for all register and interconnect assignments if* $TM(M_i) = 1$ *(i.e. the operation assigned to* M_i *is an essential concurrent operation) and* $I_{M_i} \cap O_{M_i} \neq \phi$.

Proof:

A necessary condition for a register R to be a CBILBO is that it is a self-adjacent register. For there to exist a CBILBO for all possible register assignments given a module assignment, there has to exist a module M_i such that $I_{M_i} \cap O_{M_i} \neq \phi$ (Lemma 1).

Assuming that $I_{M_i} \cap O_{M_i} \neq \phi$ we will prove that $TM(M_i) = 1$ is necessary and sufficient.
(If):
If $TM(M_i) = 1$, it has only one output variable and the register to which it gets assigned will be the only SA for the module. Since it also an input variable the same register will be the only TPG for one of the input ports of the module. Hence a CBILBO would be essential for module M_i.
(Only if):

Case(i): $TM(M_i) > 2$ More than two operations are assigned to M_i. Since each operation has a distinct output variable, therefore $| O_{M_i} | > 2$. Each of these output variables can be assigned to a separate register resulting in more than 2 output registers for M_i. From Lemma 2, a BIST embedding without a CBILBO can be found in this case.

Case(ii): $TM(M_i) = 2$ Exactly two operations are assigned to M_i. Hence $| O_{M_i} | = 2$. the two output variables can be assigned to two separate registers, say R_1 and R_2. In the worst case both output variables also belong to I_{M_i}. If R_1 and R_2 are connected to the same input port, a CBILBO is not required since one can be a TPG for that input port and the other can be an SA for the module. If they are connected to different input ports, one of the input variables which is not an output variable can be assigned to a register R_3. R_3 will serve as a TPG for one of the input ports and the self-adjacent register connected to this port (R_1 or R_2) can act as an SA precluding a CBILBO. □

If a module assignment has the property stated in Theorem 3, the register assignment and interconnect assignment cannot avoid a CBILBO. If the register and interconnect assignment is performed without regard for functional area but with the sole objective of minimizing CBILBOs, the number of CBILBOs would depend on the number of essential concurrent operations with overlapping input and output variable sets.

Definition 6 *The* **essential concurrent operation set** V_{ess} *of a module assignment is a maximal set of operations that exist in all the maximal concurrent operation sets of the module assignment. The set of modules corresponding to the operations in* V_{ess} *shall be denoted by* M_{ess}.

The essential concurrent operation set associated with a module assignment is thus the set of *all* essential concurrent operations of that module assignment.

Definition 7 *The* **storage concurrency** *of a set of variables Q, $C_S(Q)$ is the maximum number of variables in Q alive at the same time.*

Theorem 4 *Given a scheduled DFG, $G = (V, E)$ and a module assignment $\Pi_M = \{M_1, M_2, ..., M_m\}$, the lower bound on the number of CBILBOs required to test all the modules in the data path is given by*

$$C_S(\bigcup_{\forall M_i \in M_{ess}} (I_{M_i} \cap O_{M_i}))$$

Proof:

$$Let\ C_S(\bigcup_{\forall M_i \in M_{ess}} (I_{M_i} \cap O_{M_i})) = n.$$

Assume that there exists a register assignment such that all modules can be tested with n' CBILBOs and $n' < n$. From Theorem 3, only modules with a temporal multiplicity of 1 require CBILBOs. The set of modules M_i such that $TM(M_i) = 1$ is the set M_{ess}, corresponding to the essential concurrent operation set as per Definition 6. The other condition for each module M_i to require a CBILBO is $I_{M_i} \cap O_{M_i} \neq \phi$ and the register to which the variable from $(I_{M_i} \cap O_{M_i})$ gets assigned forms the CBILBO. Since we have assumed that n' CBILBOs can test all modules, therefore the variables in $(I_{M_i} \cap O_{M_i})$ for all modules in M_{ess} can be assigned to n' registers. But the storage concurrency of these variables is n and hence there is a contradiction. Therefore *at least* n CBIlBOs are required for any register assignment following the given schedule and module assignment. □

Consider the scheduled DFG shown in Fig. 5(a). Note that the output and one input variable of operations '$+_3$' and '$+_2$' is the same, namely, a and d. This occurs in the case of iterative computations. For scheduling purposes the loop is broken. Consider the following three module assignments for the scheduled DFG.

1. Assignment I: (Fig. 5(b)) Operations '$+_1$', '$+_2$' and '$+_3$' are all assigned to a different module. i.e. $M_1 = \{+_1\}$, $M_2 = \{+_2\}$ and $M_3 = \{+_3\}$. Since $I_{M_2} \cap O_{M_2} = \{d\} \neq \phi$ and $I_{M_3} \cap O_{M_3} = \{a\} \neq \phi$, therefore according to Lemma 1, registers to which variables a and d will be assigned will be self-adjacent registers. There is only one maximal concurrent operation set $V_{C_{max}} = \{+_1,+_2,+_3\}$ and each of the operation is an essential concurrent operation. However the intersection of the input and output variable sets is non-empty only for two of the three modules to which these essential concurrent operations are assigned, namely M_2 and M_3. The lower bound on the number of CBILBOs according to Theorem 4 is $C_S(\{a, d\}) = 2$ since both variables are alive at the same time.

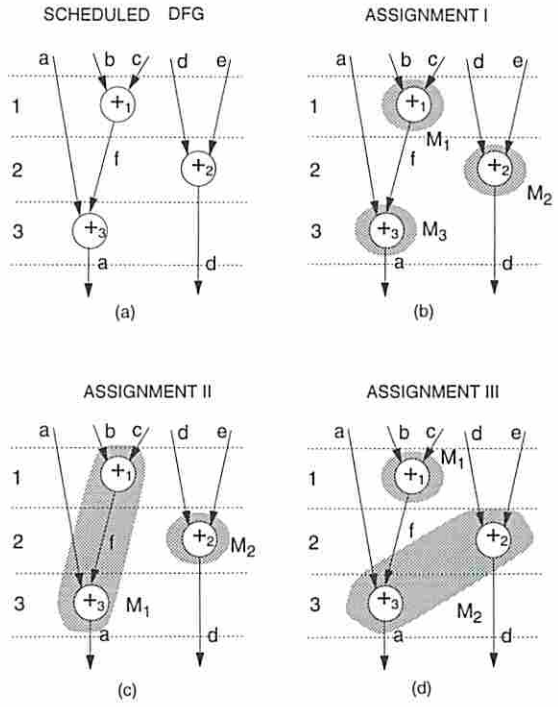

Figure 5: Module assignments and $LB_{\#CBILBOs}$

2. Assignment II: (Fig. 5(c)) In this assignment the essential concurrent operation '$+_3$' in Assignment I is assigned to the same module as operation '$+_1$'. Now we have $M_1 = \{+_1,+_3\}$ and $M_2 = \{+_2\}$. This module assignment has only one essential concurrent operation, namely, operation '$+_3$'. Since for module M_2 to which this essential concurrent operation is assigned, $I_{M_2} \cap O_{M_2} = \{d\} \neq \phi$ therefore according to Theorem 4, the lower bound on the number of CBILBOs is $C_S(\ \{d\}\ ) = 1$. Note that, $I_{M_1} \cap O_{M_1} = \{a, f\} \neq \phi$. So registers to which variables a and f will be assigned will be self-adjacent registers. Note that assignment of variable f results in a self-adjacent register only because '$+_1$' and '$+_3$' are assigned to the same module. Also even though M_1 has self-adjacent registers it does not require a CBILBO as $TM(M_1) = 2$.

3. Assignment III: (Fig. 5(d)) In this assignment we have $M_1 = \{+_1\}$ and $M_2 = \{+_2,+_3\}$. This module assignment also has only one essential concurrent operation like in Assignment II, namely, operation '$+_1$'. However for the module M_1 to which this essential concurrent operation is assigned , $I_{M_1} \cap O_{M_1} = \phi$. Hence the lower bound on the number of CBILBOs is 0. Note that $I_{M_2} \cap O_{M_2} = \{a, d\}$ so registers to which a and d are assigned will be self-adjacent registers but a register assignment can be found such that M_2 does not require a CBILBO.

Thus, to lower the number of CBILBOs in the data path the number of essential concurrent operations that have intersecting input and output variable sets should be reduced. The lower bound can be achieved by relaxing the functional area constraint. A register assignment that achieves this lower bound could have more than the minimum number of functional registers required.

4 Use of lower bound estimation in synthesis algorithms

The ability to predict area-performance characteristics of designs without actually synthesizing them is vital to produce quality designs in a reasonable time. Using a high-level synthesis system, a designer often needs to repeat the synthesis process several times while searching for a satisfactory design. Comparison of synthesis results using different module sets, module assignments and schedules are made to locate the desired design. Computation of lower bounds on *test resources* provides a synthesis system with a quick way of evaluating the testability overhead of the design. More specifically, the proposed lower bounds can be used for the following.

1. To select schedules from a set of schedules with the same latency and resource requirement. For a given behavior different schedules are possible such that they have a desired latency and satisfy a constraint on the number of modules. However the minimum number of test resources required to test the designs synthesized using these schedules could vary significantly. The lower bounds on the test resources can be used to select an appropriate schedule.

2. Given a schedule, to find a module assignment that requires few test resources. For a given schedule, different module assignments have different lower bounds on test resources. The lower bound estimation technique can be used to compare different module assignments for the same schedule. Alternatively, the lower bound estimation can be done incrementally *during* module assignment using information from partial module assignments. For example, if at some stage in the module assignment process there is a choice between many assignments, then an assignment that results in a maximal concurrent operation set with a lower output storage concurrency should be chosen.

3. To trade-off latency for area. Sometimes latency is increased if a reduction in the number of modules is desired. In some cases, increasing the latency by a few clock steps does not reduce the module requirement. However it does increase the number of different possible schedules, and thus a schedule that has a small lower bound on the number of test resources may be identified. Such a schedule could lead to savings in area.

4. To prune the search space and direct the search during register and interconnect assignment towards low testability overhead designs. The information used in the estimation of the lower

bounds (e.g. maximal concurrent operation sets and storage concurrency) can be used to prune the search space and select assignments that will achieve those bounds or will be close to the bounds.

5. To provide an independent measure of comparing the performance of different high-level synthesis systems and algorithms that perform testability optimization. High-level synthesis systems that have testability overhead as an optimization criterion use a variety of heuristics and cost functions in their algorithms. The bounds provide a common base to evaluate the quality of the various synthesis algorithms.

We have developed efficient algorithms for quick computation of the lower bounds. The algorithm for computing lower bounds on the number of TPGs and SAs has worst case complexity $O(L \cdot m \cdot (\frac{n}{m})^m)$, where L is the latency of the schedule, n is the number of operations (nodes in the DFG) and m is the number of modules assigned. The algorithm is efficient in spite of the exponential complexity because the number of modules m is usually small. Furthermore, properties of module assignments and maximal concurrent operation sets can be used to reduce the size of the exponential space. For example, consider two maximal concurrent operation sets $V^1_{C_{max}}$ and $V^2_{C_{max}}$. If all the operations in $V^1_{C_{max}}$ are scheduled in the same control step, then it can be shown that $C_{OS}(V^2_{C_{max}}) \leq C_{OS}(V^1_{C_{max}})$. Hence maximal concurrent operation sets such as $V^1_{C_{max}}$ can be ignored during the lower bound computation. The lower bound on CBILBOs can be computed in $O(L \cdot m)$ time.

5 Experimental results

In order to demonstrate the use of the proposed lower bound computation in evaluating the testability qualities of schedules and module assignments, we applied it to some well-known high level synthesis benchmarks: 1) the 2nd order differential equation - *diffeqn* [16], 2) the Tseng data flow graph - *Tseng* [17], 3) the auto regression filter element - *AR Filter* [18], and 4) the 5th order elliptic wave filter - *ewf* [19].

Table 1 depicts bounds for three different schedules of *diffeqn*. The minimum latency for this benchmark is 4. As-late-as-possible (ALAP) scheduling and as-soon-as-possible (ASAP) scheduling are two popular scheduling techniques for achieving minimum latency schedules. Both ASAP and ALAP schedules of the *diffeqn* require 5 modules. In Table 1 schedule S_1 is the ALAP schedule and schedule S_2 is the ASAP schedule. It can be seen that the lower bound on SAs, TPGs and CBILBOs required if schedule S_2 is used is lower than the lower bounds of schedule S_1. The bounds for an intermediate schedule, S_3, with the same latency are also shown. Three more schedules each

Table 1: Lower bounds for minimum latency schedules of *diffeqn*

Schedule	Type of schedule	Latency (L)	Number of modules (m)	$LB_{\#SAs}$	$LB_{\#TPGs}$	$LB_{\#CBILBOs}$
S_1	ALAP	4	5	3	5	1
S_2	ASAP	4	5	2	4	0
S_3	Intermediate	4	5	3	4	0

Table 2: Lower bounds for *diffeqn*

Schedule	Latency (L)	Number of modules (m)	$LB_{\#SAs}$	$LB_{\#TPGs}$	$LB_{\#CBILBOs}$
S_1	6	4	2	4	1
S_2	6	4	2	4	0
S_3	6	4	1	4	0

with a latency of 6 and using 4 modules are shown in Table 2. These results demonstrate that schedules that are equally attractive from the functional resources and latency point of view can differ greatly in the minimum test resource requirement.

The *Tseng* benchmark does not have any variable that is an input as well as an output variable of the same operation. Hence according to Theorem 3, the lower bound on CBILBOs is zero. We investigated the lower bounds on SAs and TPGs for all possible schedules and all possible module assignments associated with each schedule for this benchmark. Table 3 shows the lower bounds on SAs and Table 4 shows the lower bounds on TPGs. Each entry in the tables corresponds to the minimum lower bound among all possible schedules and module assignments for that particular latency and number of modules. For example, among all module assignments using 6 modules that were possible for different schedules of latency 5, the minimum lower bound on the number of SAs was 3. A '$-$' entry indicates that no schedule and module assignment solution is possible for that (L, m) value of the DFG. It can be observed that the bounds increase as the number of modules increases. Note that the *actual* functional area corresponding to the modules is not being considered here. The actual functional area depends on the particular module assignment and a higher number of modules does not necessarily imply a larger functional area [20]. The lower bounds have a strong corelation to the *number* of modules. The lower bounds on the test resources increase with number of modules because there are more modules to test which results in a larger

Table 3: Variation in SA lower bounds for *Tseng*

	Number of modules (m)							
Latency (L)	1	2	3	4	5	6	7	8
4	–	–	1	2	2	3	3	4
5	–	1	1	1	2	3	3	4
6	–	1	1	1	2	2	2	4
7	–	1	1	1	2	2	2	4
8	1	1	1	1	2	2	2	4

Table 4: Variation in TPG lower bounds for *Tseng*

	Number of modules (m)							
Latency (L)	1	2	3	4	5	6	7	8
4	–	–	2	3	3	4	5	5
5	–	2	2	3	3	4	5	5
6	–	2	2	3	3	4	4	5
7	–	2	2	3	3	4	4	5
8	2	2	2	3	3	4	4	5

number of input and output variable sets, while the storage concurrency of the variables remains the same. Tables 3 and 4 demonstrate that among two module assignments Π_M^1 and Π_M^2 such that $\mid \Pi_M^2 \mid < \mid \Pi_M^1 \mid$, assignment Π_M^2 might be desirable from the test resources point of view even if $Area(\Pi_M^2) > Area(\Pi_M^1)$.

Table 5 shows the bounds for different module assignments of the ASAP and ALAP schedules for the *AR Filter* benchmark. The latency of both schedules is 8 and the minimum number of modules for this latency is 12. All module assignments in Table 5 use 12 modules. Table 6 shows the bounds for different module assignments for a schedule of the *ewf* benchmark. The latency of the schedule used is 19. These results show that a significant variation in test resources exist for different module assignments of the same schedule as well as different schedules of the same latency.

The lower bound estimates can be used to compare the quality of the synthesized designs in terms of BIST resources. The closer the number of BIST resources are to the lower bounds, the better the quality of the synthesis algorithms in synthesizing low BIST overhead designs. Table 7

Table 5: Lower bounds for *AR Filter*

Schedule	Latency (L)	Module assignment	Number of modules (m)	$LB_{\#SAs}$	$LB_{\#TPGs}$	$LB_{\#CBILBOs}$
S_1 ALAP	8	M_1	12	5	16	0
		M_4	12	4	12	0
		M_3	12	2	8	0
S_2 ASAP	8	M_1	12	5	12	0
		M_2	12	3	8	0
		M_3	12	2	8	0

Table 6: Lower bounds for *ewf* ($L = 19$)

Module assignment	Number of modules (m)	$LB_{\#SAs}$	$LB_{\#TPGs}$	$LB_{\#CBILBOs}$
M_1	8	3	6	2
M_2	8	2	4	1
M_3	8	1	3	0

compares the BIST resources required for design synthesized by the approach in [12] with the lower bounds. *ex2* is a DFG taken from [21]. *Tseng1* and *Tseng2* are different module assignments of the *Tseng* benchmark. The lower bound on test resources was achieved in most of the cases which indicates that the synthesis algorithm performed well in optimizing test resources and that the proposed bounds are achievable *even when functional area constraints are imposed.*

6 Conclusions

Estimation of resources is important to guide high-level synthesis algorithms towards optimal solutions. Lower bounds on resources reduce the size of the search space and also provide a metric for evaluating the quality of the synthesized design. In this paper we have derived tight lower bounds on *test resources* that would be required to test the synthesized data path using partial intrusion BIST. Given complete flexibility in the assignment of registers and interconnect, the lower bounds can be achieved. The bounds give a mechanism for comparing the quality of area-performance competitive schedules and module assignments with respect to test resource requirement. The

Table 7: Comparison of actual test resources with lower bounds

DFG	# TPGs	$LB_{\#TPGs}$	# SAs	$LB_{\#SAs}$	# CBILBOs	$LB_{\#CBILBOs}$
ex2	4	4*	3	3*	1	0
Tseng1	5	5*	4	3	1	0
Tseng2	3	3*	2	2*	0	0*
diffeqn	3	3*	2	2*	1	1*

* Lower bound achieved

lower bounds along with a library of test register modules can give an estimate of the actual test area overhead. The theory on *test resource* bounds can be used in conjunction with the theory for estimating *functional resources* which has been studied extensively in literature. The *total* area of the synthesized design including test area overhead can thus be estimated accurately.

References

[1] R. Jain, A.C. Parker, and N. Park. Predicting System-Level Area and Delay for Pipelined and Non-pipelined Designs. *IEEE Trans. on CAD*, (11):955–965, August 1992.

[2] M. Rim and R. Jain. Estimating Lower-Bound Performance of Schedules Using a Relaxation Technique. In *Proc. Intn'l Conf. Comp. Design*, pages 290–294, October 1992.

[3] Y. Hu, A. Ghouse, and B.S. Carlson. Lower Bounds on the Iteration Time and the Number of Resources for Functional Pipelined Data Flow Graphs. In *Proc. Intn'l Conf. Comp. Design*, pages 21–24, October 1993.

[4] A. Sharma and R. Jain. Estimating Architectural Resources and Performance for High-Level Synthesis Applications. *IEEE Trans. on VLSI Systems*, 1(2):175–190, June 1993.

[5] S. Chaudhari and R.A. Walker. Computing Lower Bounds on Functional Units before Scheduling. In *Proc. 7th Intn'l Symp. on High-level Synthesis*, pages 36–41, May 1994.

[6] S.Y. Ohm, F.J. Kurdahi, and N. Dutt. Comprehensive Lower Bound Estimation from Behavioral Descriptions. In *Proc. Intn'l Conf. Computer-Aided Design*, pages 182–187, October 1994.

[7] 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.

[8] 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.

[9] 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.

[10] 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, November 1993.

[11] I.G. Harris and A. Orailoglu. SYNCBIST:SYNthesis for Concurrent Built-In Self-Testability. In *Proc. Intn'l Conf. Comp. Design*, pages 101–104, October 1994.

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

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] R. Jain, M. Mlinar, and A. Parker. Area-time Model for Synthesis of Non-pipelined Designs. In *Proc. Intn'l Conf. on Computer-Aided Design*, pages 48–51, Nov. 1990.

[19] S.Y. Kung, H.J. Whitehouse, and T. Kailath. *VLSI and Modern Signal Processing.* Prentice-Hall, 1985.

[20] K. Kucukcakar and A. Parker. Data Path Trade-offs using MABAL. In *Proc. 27th Design Automation Conf.*, pages 511–516, June 1990.

[21] 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.