Chapter 3

Proposed Approaches

High-level built-in self-test (BIST) synthesis systems target to generate data paths and controllers with BIST capability. The assignments of system registers, interconnections, and BIST registers greatly affect resultant built-in self-testable data paths. To reduce the complexity involved in the assignment process, in general, the three tasks are performed separately and heuristics are applied to each task to reduce BIST overhead.

We have developed three methods for high-level BIST synthesis. The first method is based on a heuristic that allocates signature registers first to achieve *k*-test session and then shares the signature registers with other registers [40]. The second one employs an integer linear program (ILP) approach that gives optimal solutions in terms of BIST area overhead, but the processing time is long [41]. The third method aims to reduce the processing time based on a heuristic. It divides a given DFG into smaller regions and then applies ILP formulation to the regions successively [42]. The three methods are described in detail in this chapter.

3.1 Heuristic Based Approach

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

In this section, we present our method that is intended to correct the problem. Our method is concerned with register assignment to embed the parallel BIST structure for a given data flow graph *in which scheduling and module assignment have been completed*. To explore a large design space, our method tries to find an optimal register assignment for each *k*-test

session, where k = 1, 2, ..., N. A register assignment for a k-test session guarantees that the synthesized circuit can be tested in k-test session. Therefore, our method offers a range of designs with different figures of merit in area and test time, and it enables a designer to select an appropriate design for his/her needs.

3.1.1 Overall Approach

The key idea of our approach is to explore a large design space by considering all the possible test sessions. The upper bound of the number of possible test sessions is equal to the total number of modules, N, which is usually small enough to make our method computationally tractable. Our method tries to find an optimal register assignment (which incurs the least area overhead) for each test session k, where k=1,2, ..., N. To find an optimal register assignment for each test session is computationally intensive. As stated in Section 2.1.2, we explained that allocation of signature registers, not test pattern generators, determines the number of test sessions. Thus, we consider the allocation of only signature registers in the first phase. Our method merges output variables (which are candidates of signature registers) to achieve a k-test session. In the next phase, our method merges the remaining variables (input variables and unallocated output variables) to maximize sharing of the allocated signature registers with other registers. After the entire signature registers are allocated and possibly shared with other registers, our method identifies candidate registers for test pattern generators and optimizes the design to reduce multiplexers and interconnections.

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

3.1.2 Source and Destination Module

In this section, we define source and destination modules that are used in the heuristic based approach. Suppose that a set of variables $\{v_1, v_2, v_3, ...\}$ is assigned to register R_i. Register R_i can be considered as a set whose elements are $v_1, v_2, v_3, ...$, i.e., $R_i=\{v_1, v_2, v_3,...\}$. We define the following two terms on register R_i.

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

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

The terms are illustrated for the following register assignments of the variables in Fig. 3.1 (a). (Note that it is a minimum clique partition.)

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

The resultant data path logic under the register assignment is given in Fig. 3.1 (b). For the register assignment of R_2, $SM(R_2)=\{SM(b), SM(e), SM(g)\}=\{M_1, M_2\}$, and $DM(R_2) =\{DM(b), DM(e), DM(g)\}=\{M_1, M_1, M_2, M_3\}=\{M_1, M_2, M_3\}$.

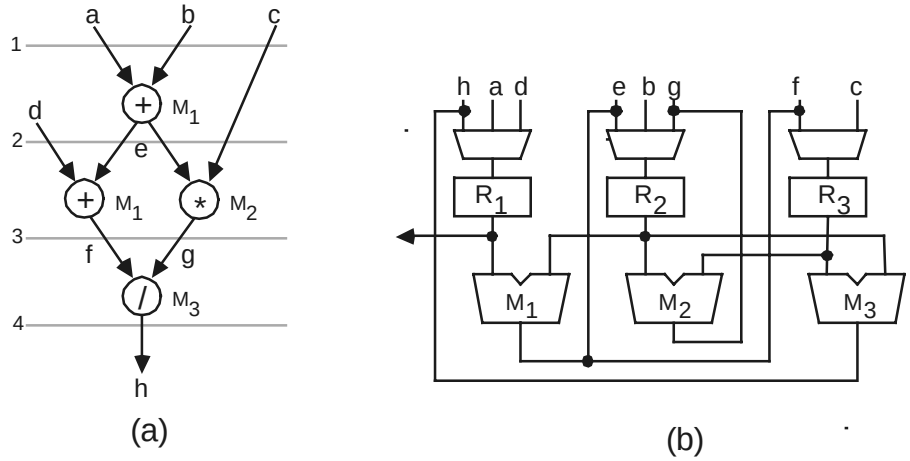


Figure. 3.1 Example data flow graph and its data path

3.1.3 Phase I: Allocation of Signature Registers

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

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

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

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

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

Allocation of signature registers is the process of merging output variables and of assigning registers to merged output variables, while meeting the three conditions described above. The goal of the allocation is to allocate the *least* number of signature registers with the *least* number of output variables assigned for each register. The intention of assigning the least number of output variables to each register is to increase the number of the unassigned output variables, which may be shared with other variables in Phase II. Thus, two output variables with

the same source module should not be merged and assigned to the same register in this phase. The condition is specified in the following:

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

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

Consider the data flow graph given in Fig. 3.2(a). The three operators, +, *, and /, are binded to M_1, M_2 and M_3, respectively. The incompatibility and compatibility graphs of output variables and their associated source modules are shown in Fig. 3.2(b) and Fig. 3.2(c), respectively.

From the compatibility graph of the output variables in Fig. 3.2(c), we look for cliques with size *k* for a *k*-test session, but any two vertices of a clique should not have the same module (refer to condition (iv)). For *k*=1, each vertex is a clique. Hence, there are four cliques, {d}, {e}, {f}, and {g}. For a 2-test session, i.e., *k*=2, there are {d,e}, {d,f}, {e,g}, and {f,g}. When *k* is three, there is no such clique. Note that the cliques {d,e,g} and {d,f,g} have two vertices, d and g, whose source module is the same. This means that a 3-test session is not optimal for the circuit. This can readily be seen from the incompatibility graph in Fig. 3.2(b). The maximum clique size of the graph is two. Hence, there are at least two signature registers available for the circuit. Clearly, testing three modules in three test sessions is not optimal when two signature registers are available.

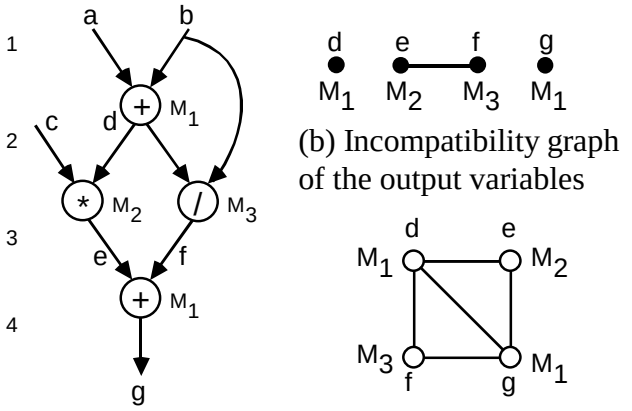


Fig. 3.2. A data flow graph with module assignment

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

For the previous example with *k*=2, all four cliques, {d,e}, {d,f}, {e,g}, and {f,g}, have the same number (one) of common neighbors. Suppose we choose clique {d,e}. The clique and all the associated edges, d-f, d-g, and e-g, are removed. As module M_1 is covered by clique {d,e}, we also remove vertex g and its associated edge, f-g. Thus, the resultant graph has only vertex f. Vertex f is selected to cover M_3 in the next process. All possible mergers for the data graph are given in Table 3.1.

The output variables of a group[1] (denoted as a circle) in Table 3.1 are merged (possibly with other input and output variables in Phase II) and assigned to a signature register. The number of groups for a merger is the number of signature registers to be allocated. For example, a merger {d,e} and {f} for k=2 results in the allocation of two signature registers. The source modules of the variables in a group are tested sequentially using the same register. However, they can be tested simultaneously with source modules of the variables belonging to a different group. For example, source module M_1 of variable d and M_2 of variable e for Merger I under k=2 are tested sequentially, but either M_1 or M_2 can be tested simultaneously with module M_3 of variable f.

Table 3.1. Possible optimal mergers for the data flow graph in Fig. 3.1(a)

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

3.1.4 Phase II: Sharing of Signature Registers with Other Registers

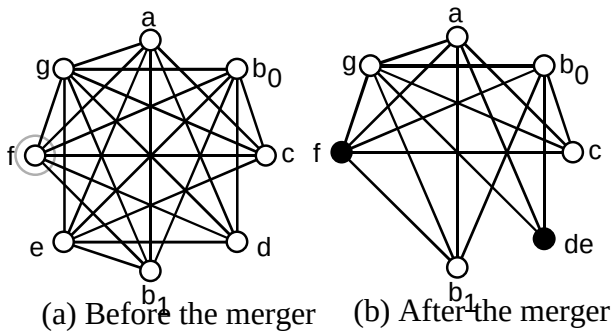
In Phase II, we merge the output variables identified in Phase I with the remaining compatibility graph of input and output variables of the data flow graph. Two steps are involved in Phase II. First, all output variables of each group identified in Phase I are merged. Next, the merged output variables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

3.1.4.1 Merger of Output Variables

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

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

We illustrate the process using the data flow graph in Fig. 3.2(a). Suppose that we have chosen Merger I for k=2 in Table 3.1. The groups of output vertices to be merged are {d,e} and {f}. (In fact, group {f} has one vertex and no need for merging.) The compatibility graph of the data flow is shown in Fig. 3.3(a). Note that delayed variable b is split into two variables, b_0 and b_1. As vertices d and e are merged, three non-common edges, c-e, b_1-e, and f-d, are removed. Then all the edges between the two groups {d,e} and {f} are deleted. Edge d-f is deleted (In fact, it has already been deleted.). The resultant compatibility graph of variables is shown in Fig. 3.3(b). The filled circles in the figure denote variables assigned to signature registers.



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

Fig. 3.3. Compatibility graphs before and after the merger

3.1.4.2 Merger of Remaining Variables

After the output variables for a k-test session are merged, all the remaining compatible vertices (i.e., input and output variables) are to be merged. The goal of the merger is to minimize the number of resultant vertices in the graph, equivalently the number of allocated registers. The process is minimum clique partition. A minimum clique partition is NP-complete, and, hence, a

heuristic is necessary for the process. In our case, if there are multiple minimum clique partitions, we need to select a partition, which is likely to incur the least area overhead. (Note that all partitions require *k*-test session.) In the following, we describe guidelines, which aim to lead the process toward a minimum clique partition and for the least area overhead.

Our method is to merge a pair of compatible vertices iteratively until no further merger is possible. The guidelines to choose a pair of compatible vertices to be merged are given in order of descending priority. A guideline *i* is applied only when its immediate guideline *i*-1 (equivalently all the previous guidelines) is tied. The guidelines are illustrated for the compatibility graph in Fig. 3.3(b). The first two guidelines aim to reduce the total number of registers for the circuit by sharing as many variables as possible. The rest of the guidelines are for the reduction of test registers and/or associated hardware, specifically multiplexers and interconnections.

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

This guideline aims to merge a pair of vertices which belong to a maximum clique of the graph. As an illustration, the number of common neighbors for each compatible pair for the compatibility graph Fig. 3.3(b) is given in Table 3.2.

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

Table 3.2: The number of common neighbors for the graph in Fig. 3.3(b)

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

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

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

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

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

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

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

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

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

Guideline 5: Choose a pair with the <u>smallest</u> number of source modules.

Since a signature register is assigned to each module in Phase I, there is no need for further sharing of output registers at this stage. This guideline aims to reduce the number of multiplexers by reducing the sharing of output registers. In the previous example, both the pairs, {a,g} and {b_0,g}, have one source module, M_3. Hence, vertex pair {a,g} is chosen arbitrarily and merged. The resultant graph is shown in Fig. 3.4. The above procedure repeats for the resultant graph to merge the next pair of vertices.

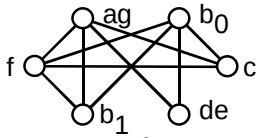


Fig. 3.4. Merger of vertices a and g

3.1.5 Post Processing

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

When all TPGs are allocated, we try to minimize multiplexers and interconnections using the methods suggested in [56]. The methods include exchanging input ports and rearranging multiplexers.

3.2 ILP Based Approach

Our method is concerned with register and interconnection assignments to embed the parallel BIST structure for a given DFG in which scheduling and module assignment have been completed. The proposed ILP formulations are for BIST register and interconnection assignments. The BIST register assignment includes system register assignment since any register that is not assigned to a BIST register (such as SR, TPG, BILBO, and CBILBO) should be a system register. The ILP formulations are solved to minimize an objective function for each *k*-test session, in our case hardware area in terms of the number of transistors.

While existing BIST synthesis methods perform system register assignments, BIST register assignments, and interconnection assignments sequentially (at the cost of global optimality), our method performs the three assignment tasks concurrently through exhaustive search. Hence, our method finds an optimal (in area) BIST design for each *k*-test session where *k* is 1, 2, ...N and N is the number of modules of the DFG. In other words, our method generates N optimal BIST designs, where each design tests the entire modules in *k* sub-test sessions.

In the following, we present ILP formulations for BIST register assignments and interconnection assignments, multiplexer assignments, which are derived directly from interconnection assignments. We also describe the objective function for our ILP model and discuss a method to reduce the search space.

3.2.1 Interconnection Assignment

Interconnections are not assigned explicitly but are determined directly from register (and module) assignments. In this section, we describe constraints for register assignments in perspective of interconnections.

One of the constraints for the proposed BIST synthesis is not to add extra paths dedicated for testing. (Refer to Section 2.1.2.) A straightforward minimization of a cost function may

create adverse paths in the synthesized data path. In order to prevent such adverse paths, it is necessary to impose some constraints as described below.

An interconnection from a register *r* to an input port *l* of a module *m* is allowed (if and) only if there exists an edge between at least one variable *v* assigned to the register and at least one operation *o* assigned to the module. In other words, a binary variable z_{rml} is 1 only if there is at least one edge $(v, o, l) \in E^i$ in the DFG such that *v* is assigned to *r* and *o* is assigned to *m*. Using an auxiliary binary variable z_{vroml}, the condition can be formulated as follows for non-commutative modules.

$$\sum_{v \in V^v, o \in V^o} z_{vroml} - z_{rml} \geq 0, \ \forall r \in R, m \in M, l \in I(m) \tag{3.1}$$

$$x_{vr} + x_{om} - 2 \cdot z_{vroml} \geq 0, \ \forall r \in R, m \in M, (v, o, l) \in E^i \tag{3.2}$$

Upon z_{rml}=1, Eq. (3.1) requires at least one auxiliary binary variable z_{vroml} to be 1. From Eq. (3.2), the auxiliary variable can be 1 only if $x_{vr} = 1, x_{om} = 1$, and $(v, o, l) \in E^i$. Therefore, the two equations guarantee the existence of an edge between the variable *v* and the input port *l* of the operation *o* if z_{rml}=1. It should be noted that the replacement of the auxiliary variable z_{vroml} in Eq. (3.1) with z_{rml} as shown in Eq. (3.2) fails work, since it requires every variable *v* and every operation *o* of an edge $(v, o, l) \in E^i$ to be assigned to *r* and *m*, respectively.

$$x_{vr} + x_{om} - 2 \cdot z_{rml} \geq 0, \ \forall r \in R, m \in M, (v, o, l) \in E^i \tag{3.3}$$

If a module is commutative, we should consider the mapping between the pseudoinput port *l** to the input port *l*. Hence, Eq. (3.3) should replace Eq. (3.1) for a commutative operation.

$$x_{vr} + x_{om} + s_{l*,l,o} - 3 \cdot z_{vroml} \geq 0, \ \forall r \in R, m \in M, l \in I(o), (v, o, l*) \in E^i \tag{3.4}$$

Similarly, if there is an interconnection from a module *m* to a register *r*, i.e., z_{mr}=1, the following formulas guarantee the existence of an edge from an operation *o* to a variable *v* such

that *o* and *v* are assigned to *m* and *r*, respectively. In the formulas, z_{omvr} is an auxiliary binary variable.

$$\sum_{o \in V^O,\, v \in V^V} z_{omvr} - z_{mr} \geq 0, \ \forall m \in M, r \in R \tag{3.5}$$

$$x_{om} + x_{vr} - 2 \cdot z_{omvr} \geq 0, \ \forall m \in M, r \in R, (o,v) \in E^O \tag{3.6}$$

3.2.2 Multiplexer Assignment

Like interconnections, multiplexers are not assigned explicitly but are implied through the register assignments. We describe identification of multiplexer sizes for registers and modules in the following.

Let an integer variable m_r represent the number of wires connected to an input of a register *r*. In other words, the variable represents the number of the multiplexer inputs attached at the input of a register *r*. Similarly, an integer variable m_{ml} represents the size of the multiplexer attached to an input port *l* of a module *m*. The process is formulated in (3.7) and (3.8).

$$\sum_{m \in M} z_{mr} = m_r, \ \ \forall r \in R \tag{3.7}$$

$$\sum_{r \in R} z_{rml} = m_{ml}, \ \ \forall m \in M, l \in I(m) \tag{3.8}$$

Two binary variables, m_{nr} and m_{nml}, are used to tally the total number of *n*-input multiplexers in Section 3.2.5. A binary variable m_{nr} is set to 1 if the input of a register *r* requires an *n*-input multiplexer. A binary variable m_{nml} is set to 1 if an input port *l* of a module *m* requires an *n*-input multiplexer. The following two formulas take care of the setting.

$$\sum_{n} m_{nr} = 1, \ \forall r \in R \tag{3.9}$$

$$\sum_{n} m_{nml} = 1, \ \forall m \in M, l \in I(m) \tag{3.10}$$

3.2.3 BIST Register Assignment

The proposed method tries to find an optimal BIST circuit for each *k*-test session. Two notations are used in the subsection. A variable *p* labels a sub-test session (which tests a subset of modules) performed during a *k*-test session, where $p = 1, 2, \ldots k$. A set *K* is defined as the set of sub-test sessions, which is $\{1, 2, \ldots k\}$ for a *k*-test session. We present ILP formulas for BIST register assignments in the following.

Consider a *partial* data path given in Fig. 3.5. It has two modules, M3 and M4. So the data path, when registers are properly reconfigured for BIST, can be tested in one test session or two test sessions. So $K=\{1\}$ for the *1*-test session, and $K=\{1,2\}$ for the *2*-test session.

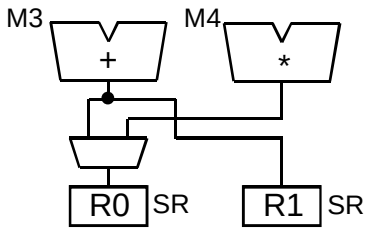


Figure 3.5 A partial data path for an SR

3.2.3.1 SR Assignment

In this subsection, we formulate the assignment of SRs for our ILP model. *All examples given in this subsection refer to the data path in Fig. 3.5.* A binary variable s_{mrp} is 1 only if a register *r* is reconfigured as the SR for a module *m* in a *p* sub-test session. Otherwise, the variable is 0. A register *r*, when reconfigured to an SR, can test a module *m* only if there is an interconnection from the module *m* and the register *r*. Hence, the following condition should hold.

$$z_{mr} - \sum_{p \in K} s_{mrp} \geq 0, \quad \forall m \in M, r \in R \tag{3.11}$$

We illustrate for the case of the *two*-test session, in which K={1, 2}. There are eight s_{mrp} variables (i.e., $s_{3,0,1}$, $s_{4,0,1}$, $s_{3,1,1}$, and $s_{4,1,1}$ for p=1 and $s_{3,0,2}$, $s_{4,0,2}$, $s_{3,1,2}$, and $s_{4,1,2}$ for p=2) to be considered. Among the eight variables, the above equation sets two variables, $s_{4,1,1}$, and $s_{4,1,2}$, to 0 since z_{41} =0. In other words, register R1 cannot be reconfigured as an SR for module M4 in either sub-test session.

The next condition is that each module should be tested only once in a sub-test session p during the entire k-test session. Thus,

$$\sum_{r \in R,\, p \in K} s_{mrp} = 1, \ \forall\, m \in M \tag{3.12}$$

From Eq. (3.12), the following set of equations are obtained for module M3 and module M4.

$$s_{3,0,1} + s_{3,1,1} + s_{3,0,2} + s_{3,1,2} = 1$$

$$s_{4,0,1} + s_{4,0,2} = 1$$

Finally, although an SR may be shared by several modules during testing, it should not be shared in the same sub-test session. Hence,

$$\sum_{m \in M} s_{mrp} \leq 1, \quad \forall\, r \in R, p \in K \tag{3.13}$$

For register R0 under the *two*-test session, Eq. (3.13) yields $s_{3,0,1} + s_{4,0,1} \leq 1$ for p=1 *and* $s_{3,0,2} + s_{4,0,2} \leq 1$ for p=2.

3.2.3.2 TPG Assignment

In this subsection, we formulate the assignment of TPGs. *All the illustrations given in this subsection refer to Fig. 3.6.* Note that the data path can be tested in a *1*-test session or a *2*-test session.

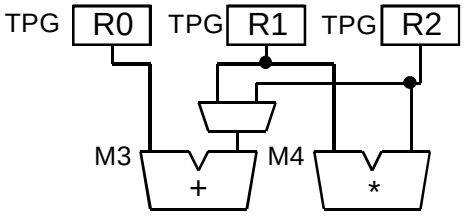


Figure 3.6 A partial data path for a TPG

A binary variable t_{rmlp} is 1 only if a register *r* is the TPG for an input port *l* of a module *m* in a *p* sub-test session. As for the case of the SR assignment, a TPG needs a connection from the register and the input of the module. Each input port of a module *m* needs only one TPG for the entire *k*-test session. The following two equations specify them.

$$z_{rml} - \sum_{p \in K} t_{rmlp} \geq 0, \quad \forall r \in R,\ m \in M,\ l \in I(m) \tag{3.14}$$

$$\sum_{r \in R,\, p \in K} t_{rmlp} = 1, \forall m \in M,\ l \in I(m) \tag{3.15}$$

From Eq. (3.14), we need to consider only ten variables, $t_{0,3,0,1}$, $t_{1,3,1,1}$, $t_{1,4,0,1}$, $t_{2,3,1,1}$, and $t_{2,4,1,1}$ for p=1 and $t_{0,3,0,2}$, $t_{1,3,1,2}$, $t_{1,4,0,2}$, $t_{2,3,1,2}$, and $t_{2,4,1,2}$ for p=2, for the data path. All other binary variables t_{rmlp} are set to 0 due to the lack of interconnections. Eq. (3.15) yields the following equations for module M3: $t_{0,3,0,1} + t_{0,3,0,2} = 1$ and $t_{1,3,1,1} + t_{1,3,1,2} + t_{2,3,1,1} + t_{2,3,1,2} = 1$.

All TPGs assigned to the inputs of a module and the SR of the module should be active in the same sub-test session as specified below.

$$\sum_{r \in R} t_{rm0p} - \sum_{r \in R} t_{rm1p} = 0, \ \forall m \in M, \ p \in K \tag{3.16}$$

$$\sum_{r \in R} s_{mrp} - \sum_{r \in R} t_{rm0p} = 0, \ \forall m \in M, \ p \in K \tag{3.17}$$

For example, $t_{0,3,0,1} - (t_{1,3,1,1} + t_{2,3,1,1}) = 0$ for module M3 under p=1 from Eq. (3.16), and $s_{3,0,1} + s_{3,1,1} - t_{0,3,0,1} = 0$ from Eq. (3.17).

Finally, a TPG should not be shared between the two input ports of a module. This requirement is necessary to achieve high fault coverage. Thus,

$$t_{rm0p} + t_{rm1p} \leq 1, \ \forall r \in R, \ m \in M, \ p \in K \tag{3.18}$$

3.2.3.3 BILBO and CBILBO Assignment

To formulate the BILBO and CBILBO assignment, we need to define a binary variable x. A binary variable x is 1 when $\sum_i x_i$ is at least 1 and is 0 otherwise where x_i is a binary variable. In other words, the variable x is the OR operation of all x_is. A variable x is formulated as follows.

$$\bar{x}_i \cdot x - \sum_i x_i \geq 0, \text{ where } \bar{x}_i \text{ is a constant equal to the number of } x_i\text{'s.} \tag{3.19}$$

For example, let x_i, i=1, 2, 3 have values x_1=0, x_2=1, and x_3=0. Then, $\bar{x}_i$ is 3, and $\sum_i x_i$ is 1. The variable x should be 1 to satisfy Eq. (3.19). x is the 0 only if all x_is are 0.

We describe a condition that requires a register be reconfigured into a BILBO or a CBILBO first. Then, we elaborate the case where the register should be reconfigured into only a CBILBO. A binary variable t_r (s_r) is 1 only when a register r is used as a TPG (an SR) in at least one sub-test session. If both t_r and s_r are 1, then the register r should be reconfigured into a BILBO or a CBILBO. Eq. (3.20) determines if a register r is used as a TPG, and Eq. (3.21) determines if it is used as an SR. Only if the register is used as a TPG and an SR, a binary

variable b_r is set to 1 due to (3.22) and (3.23). Hence, b_r =1 indicates that the register r should be reconfigured into either a BILBO or a CBILBO.

$$\bar{t}_{rmlp} \cdot t_r - \sum_{m \in M, l \in I(m), p \in K} t_{rmlp} \geq 0, \ \forall r \in R \tag{3.20}$$

$$\bar{s}_{mrp} \cdot s_r - \sum_{m \in M, p \in K} s_{mrp} \geq 0, \ \forall r \in R \tag{3.21}$$

$$s_r + t_r - b_r \leq 1, \ \forall r \in R \tag{3.22}$$

$$s_r + t_r - 2 \cdot b_r \geq 0, \ \forall r \in R \tag{3.23}$$

In the above, we identified the case where a register should be a BILBO or a CBILBO. We refine the case next. If a register is used as a TPG and an SR in the *same* sub-test session p, the register should be reconfigured as a CBILBO. Otherwise, it should be a BILBO. First, let us check the case for a TPG. A binary variable t_{rp} is 1 if a register r is used as a TPG in a sub-test session p and is 0 otherwise. It is formulated as

$$\bar{t}_{rmlp} \cdot t_{rp} - \sum_{m \in M, l \in I(m)} t_{rmlp} \geq 0, \ \forall r \in R, p \in K. \tag{3.24}$$

Note that a binary variable t_{rmlp} is set to 1 only if a register r is the TPG for an input port l of a module m in a p sub-test session. Hence, Eq. (3.24) sets the binary variable t_{rp} to 1 if a register r is a TPG of *any* module in a sub-test session p.

Next, let us check the case for an SR. A binary variable s_{rp} is 1 if a register r is used as an SR in a sub-test session p and is 0 otherwise. The formulation is

$$\bar{s}_{mrp} \cdot s_{rp} - \sum_{m \in M} s_{mrp} \geq 0, \ \forall r \in R, p \in K. \tag{3.25}$$

Note that a binary variable s_{mrp} is 1 only if a register r is used as the SR for a module m in the p sub-test session. Hence, Eq. (3.25) sets the binary variable s_{rp} to 1 if a register r is used as an SR for *any* module in a sub-test session p.

Eq. (3.26) and (3.27) sets a binary variable c_{rp} to 1 only if t_{rp}=1 and s_{rp}=1. In other words, c_{rp} is 1 only if a register r is reconfigured into a TPG and an SR in the same sub-test session p. Hence, the register should be reconfigured into a CBILBO. Finally, a binary variable c_r is set to 1 due to Eq. (3.28) if there is at least one sub-test session which requires a register r to be reconfigured as a CBILBO. This implies that the register r should be a BILBO if c_{rp} =0 (under b_r =1).

$$s_{rp} + t_{rp} - c_{rp} \leq 0, \quad \forall r \in R, p \in K \tag{3.26}$$

$$s_{rp} + t_{rp} - 2 \cdot c_{rp} \geq 0, \quad \forall r \in R, p \in K \tag{3.27}$$

$$\bar{c}_{rp} \cdot c_r - \sum_{p \in K} c_{rp} \geq 0, \ \forall r \in R \tag{3.28}$$

3.2.4 Constants

Coefficients (such as in digital filters or signal processing applications) are usually constants. An operation in a DFG may receive constant values on an input port. If an input port of a module is connected to only one constant, it is usually hardwired to necessary constant logic values. The hardwired input port is resolved into the logic during the optimization process of a logic synthesis. So it is unnecessary to connect a TPG for the input port.

However, if an input port receives more than one constant, the input port is unlikely to be removed during the logic synthesis. Since the input port is not connected to any variable in the DFG, there is no register to be reconfigured into a TPG for the input port. To achieve a high fault coverage, it is necessary to add an extra TPG to the input port or to add an extra path to an existing TPG. Therefore, a case in which an input port is connected to only constants should be avoided in the assignment process if possible.

To avoid such a case, we introduce a new binary variable t^c_{cmlp}. The variable is 1 only if a constant *c* needs to be reconfigured into a TPG for an input port *l* of a module *m* in a *p* sub-test session. Thus, previous Eqs. (3.14) through (3.17) need to be modified as follows:

$$z_{cml} - \sum_{p \in K} t^c{}_{cmlp} \geq 0, \quad \forall\ c \in C, m \in M, l \in I(m) \tag{3.14′}$$

$$\sum_{r \in R,\, p \in K} t_{rmlp} + \sum_{c \in C,\, p \in K} t^c{}_{cmlp} = 1, \forall\, m \in M, l \in I(m) \tag{3.15′}$$

$$\sum_{r \in R} t_{rm0p} + \sum_{c \in C} t^c{}_{cm0p} - \sum_{r \in R} t_{rm1p} - \sum_{c \in C} t^c{}_{cm1p} = 0, \ \forall\, m \in M, \ p \in K \tag{3.16′}$$

$$\sum_{r \in R} s_{mrp} - \sum_{r \in R} t_{rm0p} - \sum_{c \in C} t^c{}_{rm0p} = 0, \ \forall\, m \in M, \ p \in K \tag{3.17′}$$

By assigning a high cost in setting t^c_{cmlp} to 1, the case can be avoided if possible.

3.2.5 Objective Function

The objective of an ILP is to minimize an objective function, i.e., cost function, for the ILP formulations. The cost function to be minimized for the proposed ILP model represents hardware area (in terms of transistor count) and is given below.

$$w_r\, N_r + w_s\, N_s + w_t\, N_t\ + w_b\, N_b + w_c\, N_c + \sum\nolimits_{\mathrm{n}} w_{m_n} N_{m_n}\ + w_{t^c}\, N_{t^c}$$

where N_r, N_s, N_t, N_b, N_c, and N_{m_n} are the number of system registers, SRs, TPGs, BILBOs, CBILBOs, and *n*-input multiplexers, respectively. A variable N_{t^c} is the number of constants that should be reconfigured into TPGs. Weights, w_r, w_s, w_t, w_b, and w_{m_n} are the number of transistors for the corresponding hardware. The weight w_{t^c} is the cost for constants. We assign w_{t^c} a large

number greater than any other weight to avoid, if possible, the case in which a port needs an extra TPG or an extra wire to an existing TPG.

To compute the value (cost) of the objective function, it is necessary to compute the number of individual registers, BIST registers, and multiplexers. They are readily calculated by summing up the related binary variables as shown below:

- Number of CBILBOs: $N_c = \sum_{r \in R} c_r$ (3.29)
- Number of BILBOs: $N_b = \sum_{r \in R} b_r - N_c$ (3.30)
- Number of SRs: $N_s = \sum_{r \in R} s_r - N_c - N_b$ (3.31)
- Number of TPGs: $N_t = \sum_{r \in R} t_r - N_c - N_b$ (3.32)
- Number of unconfigured system registers: $N_r = N_R - (N_s + N_t + N_b + N_c)$ (3.33)

  where N_R represents the minimum number of registers necessary for the given DFG.
- Number of constant to be reconfigured as TPG: $N_{t^c} = \sum_{c \in C, m \in M, l \in I(m), p \in K} t^c{}_{cmlp}$ (3.34)
- Number of *n*-input multiplexer: $N_{m_n} = \sum_{m \in M, l \in I(m)} m_{nml} + \sum_{r \in R} m_{nr}$ (3.35)

Note that the summation of b_r variables in Eq. (3.30) includes the number of CBILBOs, N_c. Thus, N_c is subtracted from the summation in Eq. (3.29). When summing s_r and t_r variables in Eq. (3.31) and Eq. (3.32), the numbers of CBILBOs and of BILBOs are also counted. Therefore, the two numbers are subtracted in the according equations.

3.2.6 Reduction of the Search Space

An ILP allows one to describe the problem formally and to obtain an optimal solution. However, an ILP method can be applied to small or medium range problems since the

complexity of an ILP problem grows rapidly with the problem size. Hence, it is often necessary to employ a measure to reduce the problem size, equivalently the search space. We describe our method for the reduction of the search space for the proposed ILP model.

A register assignment in high-level synthesis is to assign a subset of compatible variables into a register. Therefore, *n incompatible* variables are assigned to *n* different registers. Consider two incompatible variables, v_i and v_j, and two registers, R_k and R_l. There are two possible register assignments, ($v_i \rightarrow R_k$, $v_j \rightarrow R_l$) and ($v_i \rightarrow R_l$, $v_j \rightarrow R_k$). The two assignments are isomorphic for our ILP model. Hence, an arbitrary selection of the two assignments does not affect the quality of the solution, but it reduces the search space by half. For the case of *n* incompatible variables, an arbitrary assignment of the variables into *n* registers reduces the search space by $n!$.

We employ the above method for our ILP model to reduce the search space. We scan each control step to identify the maximal horizontal crossing of the DFG. Then we assign all the variables (that are incompatible) on the control step to registers arbitrarily. The method is simple yet highly effective.

3.3 Region-Wise ILP Based Approach

In this section, we present ILP formulation for high-level BIST synthesis that implements our proposed region-wise optimal heuristic. Our method is concerned with shortening the processing time of the ILP approach presented in the previous section.

3.3.1 Motivation and Overall Procedure

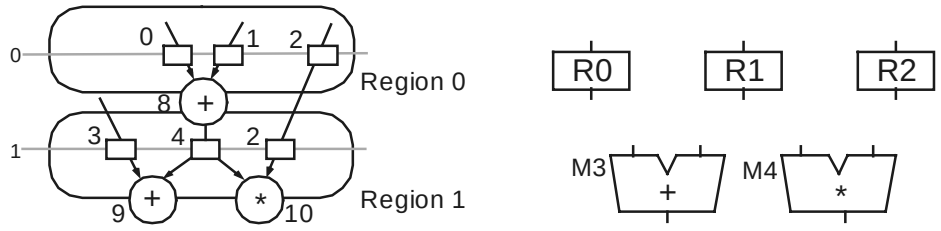
Existing BIST synthesis methods perform three tasks, system register assignment, interconnection assignment, and BIST register assignment, sequentially (at the cost of global optimality). Our method based on an ILP [41] performs the three assignment tasks concurrently to yield optimal design. However, our ILP method involves a long processing time due to the inherent characteristic of an ILP exhaustive search. Hence, it is limited to small and medium size problems and we proposed a heuristic to address the problem. The heuristic first divides a given DFG into smaller regions and then applies ILP formulation to the regions successively. As our heuristic divides the problem space into smaller pieces, it reduces the overall processing time at the cost of the sub-optimal solution.

The BIST synthesis process considered in this thesis is in essence register assignment. Consider (i+j) variables to be assigned to k registers. The total number of possible assignments is $k^{(i+j)}$, of which some assignments are illegal or isomorphic. Suppose that we divide the variables into two groups with i variables for one group and j variables for the other group. Let us assign the i variables of the group to k registers first and then assign the j variables of the other group to the same k registers. In this case, the total number of assignments is reduced to k^i+k^j, which, in turn, results in a shorter processing time. The proposed heuristic is to partition variables into smaller groups and to serially assign variables in each group.

The remaining issue is how to partition variables into smaller groups. In theory, variables can be partitioned arbitrarily, but the partition affects the quality of the solution. All the variables

of a control step are incompatible and should be assigned to different registers. Hence, the number of possible assignments for i variables of a control step to k registers is $k!/(k-i)!$, where $k \geq i$, which is much smaller than that of the general case. Note that the general case of assigning i variables to k registers has i^k possibilities. Based on this observation, we propose the partition of variables along control steps. A region for our heuristic contains all variables of s consecutive time steps, where s called region size is the number of control steps of a region and is a predetermined positive integer. A smaller s speeds up the processing at the cost of the compromised quality. When an operator resides at the boundary of two adjacent regions, the inputs of the operator belong to the preceding region and the output belongs to the succeeding region.

Consider the DFG in Fig. 3.7 (a). The region size s is set to 1. Note that the inputs of operator 8 belong to region 0 and the output of the operator to region 1. The maximal horizontal crossing is 3 and, hence, needs at least three registers. There are two different types of operators that require two modules. Let us suppose that operations 8 and 9 are assigned to module M3, and operation 10 to module M4. The data path resources necessary for the DFG are shown in Fig. 3.7 (b). Register and interconnection assignments for our heuristic are illustrated using the DFG in Fig. 3.7.



(a) A part of data flow graph (b) Given data path resources

Figure 3.7 Data flow graph and given data path resources

Our heuristic considers one region at a time starting from region 0. Region 0 has variables 0, 1, and 2, and each variable is assigned to R0, R1, and R2, arbitrarily. (Note that all of the other assignments are isomorphic to the assignment.) So we obtain,

$$x_{0,0} = x_{1,1} = x_{2,2} = 1.$$

The necessary interconnections for the register assignment are

$$z_{0,3,0} = z_{1,3,1} = 1.$$

In region 1, ILP equations for the register assignment are

$$\begin{aligned} x_{3,0} + x_{3,1} + x_{3,2} &= 1 \\ x_{4,0} + x_{4,1} + x_{4,2} &= 1 \\ x_{2,0} + x_{3,0} + x_{4,0} &\leq 1 \\ x_{2,1} + x_{3,1} + x_{4,1} &\leq 1 \\ x_{2,2} + x_{3,2} + x_{4,2} &\leq 1. \end{aligned}$$

Since $x_{2,2} = 1$ for region 0, $x_{3,2} = x_{4,2} = 0$.

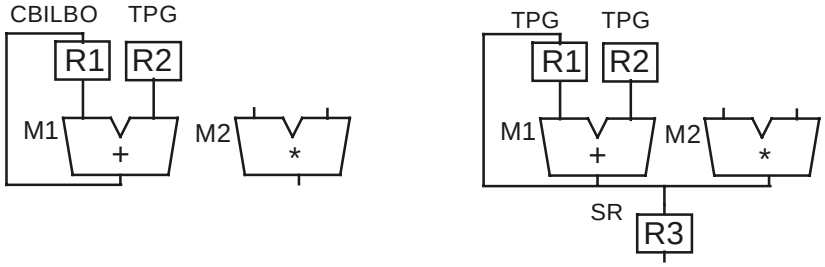
Interconnection assignments for the case of non-commutative operations are

$$\begin{aligned} x_{8,3} + x_{4,0} - z_{3,0,0} &\leq 1 \\ x_{8,3} + x_{4,1} - z_{3,1,0} &\leq 1 \\ x_{8,3} + x_{4,2} - z_{3,2,0} &\leq 1. \end{aligned}$$

The assignments for commutative operations are slightly more complex and have been omitted.

ILP equations specify a set of constraints for system register assignment, interconnection assignment, and BIST register assignment for each region. After register and interconnection assignments are made for a region, *the assignments are fixed for the processing of the remaining regions*. Therefore, registers and interconnections are added, but not removed, as more regions are processed. In processing a region *i*, the BIST register assignment attempts to construct the parallel BIST structure (which is to assign a TPG to each input port and an SR to the output port of every module) with existing, i.e., already assigned, registers and interconnections. However, unlike register and interconnection assignments, it may modify previous BIST assignments, if necessary, to improve the quality of the existing solution. Note that the attempt to construct the parallel BIST structure may not be successful in the middle of the process due to the lack of assigned registers and interconnections.

The BIST assignment process is illustrated in Fig. 3.7. After processing region *i*-1, the BIST structure for module M1 is complete, but incomplete for module M2 in Fig. 3.8 (a). After processing region *i*, register R3 and some associated interconnections are added in Fig. 3.8 (b). The BIST assignment process modifies the current assignment of register R1 from CBILBO to a TPG to reduce the hardware overhead.



(a) After processing region *i*-1 (b) After processing region *i*

Figure 3.8 Progress of BIST register assignment

The following additional nomenclatures are used for the region-wise ILP formulation described in the thesis.

- V_v^c is the set of variables in the current region *c*.
- V_o^c is the set of operations in the current region *c*.
- E_i^c is the set of ordered triples (*v*, *o*, *l*) in the current region *c*.
- E_o^c is the set of ordered doubles (*o*, *v*) in the current region *c*.
- M_i^c is the set of modules such that a region processed thus far, including the current region c, contains the **input** port(s) of an operator assigned to the module. For example, M_i^0 = {M_3} for region 0 in Fig. 3.7 (a) since operator 8 is assigned to M_3, and the input ports of operator 8 are included in region 0. Similarly, M_i^1 = {M_3, M_4} for region 1. Note that every input port of a module in M_i^c has connections to an assigned register.

- M_o^c is the set of modules such that a region processed thus far, including the current region c, contains the **output** port of an operator assigned to the module. For example, $M_o^0 = \{\varnothing\}$ for region 0 in Fig. 3.7 (a), and $M_o^1 = \{M_3\}$ for region 1. Note that the output port of a module in M_o^c has a connection to an assigned register.

In the following, we present ILP formulation for our heuristic for the interconnection assignment and the BIST assignment, and discuss the objective function for our ILP model.

3.3.2 Region-Wise System Register Assignment

The ILP formulation for system register assignment in Eq. (2.5) is modified as follows to account for our region-wise processing:

$$\sum_{r \in R} x_{vr} = 1, \ \forall \ v \in V_v^c . \tag{3.36}$$

3.3.3 Region-Wise Interconnection Assignment

Interconnections are not assigned explicitly, but are determined directly from register (and module) assignments. In this section, we describe constraints for register assignments in perspective of region-wise interconnections. The equations for the interconnection assignment in Eq. (2.8) and (2.12) are modified as follows:

$$x_{vr} + x_{om} - z_{rml} \leq 1, \ \forall \ r \in R, m \in M_i^c, (v,o,l) \in E_i^c \tag{3.37}$$

$$x_{vr} + x_{om} + s_{l^*,l,o} - 2 \cdot z_{rml} \geq 0, \ \forall \ r \in R, m \in M_i^c, l \in I(o), (v,o,l^*) \in E_i^c . \tag{3.38}$$

One of the constraints for the proposed BIST synthesis is not to add extra paths dedicated for testing (refer to Section 2.1.2). A straightforward minimization of a cost function may create adverse paths in the synthesized data path. In order to prevent such adverse paths, it is necessary to impose a constraint as described below.

An interconnection (i.e., wire) between a register and a module is allowed only if there exists an edge between a variable assigned to the register and an operation assigned to the module. In other words, a binary variable z_{rml} is 1 only if there is an edge (v, o, l) $\in E_i$ in the DFG such that v is assigned to r and o assigned to m. A non-commutative module should meet the following equations:

$$\sum_{v \in V_v^c, o \in V_v^c} z_{vroml} - z_{rml} \geq 0, \ \forall r \in R, m \in M_i^c, l \in I(m) \tag{3.39}$$

$$x_{vr} + x_{om} - 2 \cdot z_{vroml} \geq 0, \ \forall r \in R, m \in M_i^c, (v,o,l) \in E_i^c. \tag{3.40}$$

Eq. (3.39) sets at least one auxiliary binary variable z_{vroml} to 1 provided z_{rml} =1. Eq. (3.40) sets $x_{vr} = 1$ and $x_{om} = 1$, where (v, o, l) $\in$ E_i^c provided the auxiliary variable z_{vroml} = 1. Therefore, the two equations guarantee the existence of an edge between v and o and if z_{rml} =1. If the module is commutative, we have to consider the mapping between the pseudoinput l* and the input l. Hence, Eq. (3.41) should replace Eq. (3.40).

$$x_{vr} + x_{om} + s_{l^*,l,o} - 3 \cdot z_{vroml} \geq 0, \ \forall r \in R, m \in M_i^c, l^* \in I(o), (v,o,l^*) \in E_i^c. \tag{3.41}$$

It should be noted that the replacement of the auxiliary variable z_{vroml} in (3.40) with z_{rml} as shown below fails to work, since it assigns every v and o to r and m, respectively, for (v, o, l) $\in E_i$.

$$x_{vr} + x_{om} - 2 \cdot z_{rml} \geq 0, \ \forall r \in R, m \in M_i^c, (v,o,l) \in E_i^c. \tag{3.40'}$$

Similarly, auxiliary variable z_{omvr} is defined to guarantee the existence of an edge between o and v and if z_{mr} =1. Hence,

$$\sum_{o \in V_o^c, v \in V_v^c} z_{omvr} - z_{mr} \geq 0, \ \forall m \in M_o^c, r \in R \tag{3.42}$$

$$x_{om} + x_{vr} - 2 \cdot z_{omvr} \geq 0, \ \forall m \in M_o^c, r \in R, (o,v) \in E_o^c. \tag{3.43}$$

3.3.4 Region-Wise BIST Register Assignment

The proposed method tries to find a BIST circuit for each *k*-test session, where *k* is from 1 to N and N is the number of modules. Two notations are used in the subsection. A variable *p* labels a sub-test session, which tests a subset of modules of the BIST circuit, performed during a *k*-test session, where *p* = 1, 2, … *k*. A set *K* is defined as the set of sub-test sessions, which is {1, 2, …*k*} for a *k*-test session. The example data path in Fig. 3.9 can be tested in a *one*-test session (*k*=1) or a *two*-test session (*k*=2). We present ILP formulation for the region-wise BIST register assignment in the following sections.

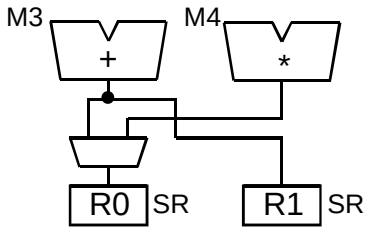


Figure 3.9 A data path for SR

3.3.4.1 SR Assignment

Suppose that the data path shown in Fig. 3.8 is currently under consideration for an SR assignment. *All examples given in this subsection refer to the data path in Fig. 3.8.* A binary variable s_{mrp} is 1 only if a register *r* is reconfigured as the SR for a module *m* in a *p* sub-test session. Otherwise, the variable is 0. A module *m* can be tested by a register *r* (which is to be

reconfigured to an SR) only if there is an interconnection between the module *m* and the register *r*. Hence, the following condition should hold:

$$z_{mr} - \sum_{p \in K} s_{mrp} \geq 0, \quad \forall m \in M_o^c, r \in R \tag{3.44}$$

where *K* is the set of sub-test sessions, i.e., $K=\{1, 2, \ldots k\}$ for a given *k*-test session. For the example data path, $K=\{1, 2\}$ for the *two*-test session. It is sufficient to consider only six s_{mrp} variables for the *two*-test session (i.e., $s_{3,0,1}$ $s_{4,0,1}$ $s_{3,1,1}$ for $p=1$ and $s_{3,0,2}$ $s_{4,0,2}$ $s_{3,1,2}$ for $p=2$). Note that the variable, $s_{4,1,p}$, is always set to 0 due to the lack of an interconnection between module M4 and register R1.

The next condition is that each module should be tested only once in a sub-test session *p* during the *k*-test session. Thus,

$$\sum_{r \in R, p \in K} s_{mrp} = 1, \ \forall m \in M_o^c. \tag{3.45}$$

From (3.45), the following equations are obtained for module M3 and module M4:

$$s_{3,0,1} + s_{3,1,1} + s_{3,0,2} + s_{3,1,2} = 1.$$

$$s_{4,0,1} + s_{4,0,2} = 1.$$

Finally, although an SR may be shared by several modules during the testing, it should not be shared in the same sub-test session. Hence,

$$\sum_{m \in M_o^c} s_{mrp} \leq 1, \quad \forall r \in R, p \in K. \tag{3.46}$$

For the register 0 under the *two*-test session, Eq. (3.46) yields as $s_{3,0,1} + s_{4,0,1} \leq 1$ for $p=1$ *and* $s_{3,0,2} + s_{4,0,2} \leq 1$ for $p=2$.

3.3.4.2 TPG Assignment

In this subsection, we formulate region-wise assignment of TPGs for our ILP model. *All of the illustrations given in this subsection refer to Fig. 3.10.* Note that the data path can be tested in a *one*-test session or in a *two*-test session.

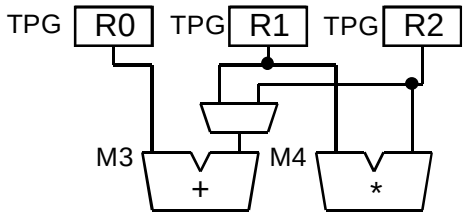


Figure 3.10 A data path for TPG

A binary variable t_{rmlp} is 1 only if a register *r* is the TPG for the l^{th} input of a module *m* in a *p* sub-test session. As in the case of the SR assignments, it needs a connection between the register *r* and the l^{th} input of a module *m*. Each input port of a module *m* needs one and only one TPG for the entire *k*-test session. The following two equations specify them.

$$z_{rml} - \sum_{p \in K} t_{rmlp} \geq 0, \quad \forall \ r \in R,\ m \in M_i^C,\ l \in I(m) \tag{3.47}$$

$$\sum_{r \in R, p \in K} t_{rmlp} = 1, \forall\, m \in M_i^C,\ l \in I(m). \tag{3.48}$$

The t_{rmlp} of binary variables, which should be considered for the example data path, are $t_{0,3,0,1}$ $t_{1,3,1,1}$ $t_{1,4,0,1}$ $t_{2,3,1,1}$ $t_{2,4,1,1}$ for *p*=1 and $t_{0,3,0,2}$ $t_{1,3,1,2}$ $t_{1,4,0,2}$ $t_{2,3,1,2}$ $t_{2,4,1,2}$ for *p*=2. Equation (3.48) yields the following equations for the example data path: $t_{0,3,0,1} + t_{0,3,0,2} = 1$ and $t_{1,3,1,1} + t_{1,3,1,2} + t_{2,3,1,1} + t_{2,3,1,2} = 1$ for the module M3.

All TPGs assigned to the inputs of a module and the SR of the module should be active in the same sub-test session as specified below.

$$\sum_{r \in R} t_{rm0p} - \sum_{r \in R} t_{rm1p} = 0, \ \forall\, m \in M_i^C, \ p \in K \tag{3.49}$$

$$\sum_{r \in R} s_{mrp} - \sum_{r \in R} t_{rm0p} = 0, \ \forall\, m \in M_i^C, \ p \in K. \tag{3.50}$$

For module M3 under p=1, $t_{0,3,0,1} - (t_{1,3,1,1} + t_{2,3,1,1}) = 0$ from Eq. (3.49) and $s_{3,0,1} + s_{3,1,1} - t_{0,3,0,1} = 0$ from Eq. (3.50).

Finally, a TPG should not be shared between the two input ports of a module. This requirement is necessary to achieve high fault coverage. Thus,

$$t_{rm0p} + t_{rm1p} \leq 1, \ \forall\, r \in R, \ m \in M_i^C, \ p \in K. \tag{3.51}$$

3.3.4.3 BILBO and CBILBO Assignment

The OR operation in ILP, to formulate BILBO and CBILBO assignment, was already described in Section 3.2.3.3. But, it will be described briefly again for convenience. A variable x is 1 when $\sum_i x_i$ is at least 1, and 0 otherwise, where x_i is a binary variable. In other words, the variable x is the OR operation of all x_is. The variable x is formulated as follows.

$$\bar{x}_i \cdot x - \sum_i x_i \geq 0, \tag{3.52}$$

where $\bar{x}_i$ is a constant equal to the number of x_is . The variable x should be 1 to satisfy Eq. (3.52). x is the 0 only if all x_is are 0.

We describe a condition that requires a register to be reconfigured into a BILBO or a CBILBO first. Then, we elaborate the case where the register should be reconfigured into only a CBILBO. A binary variable t_r (s_r) is 1 only when a register r is used as a TPG (an SR) in at least one sub-test session. If $t_r = s_r = 1$, then the register r should be reconfigured into a BILBO or a CBILBO. Eq. (3.53) determines whether a register r is used as a TPG, and Eq. (3.54) determines if it is used as an SR. If the register is used as a TPG and an SR, a binary variable b_r is set to 1

due to (3.55) and (3.56). Hence, b_r =1 indicates that the register r should be reconfigured into either a BILBO or a CBILBO.

$$\bar{t}_{rmlp} \cdot t_r - \sum_{m \in M_i^c, l \in I(m), p \in K} t_{rmlp} \geq 0, \ \forall r \in R \tag{3.53}$$

$$\bar{s}_{mrp} \cdot s_r - \sum_{m \in M_o^c, p \in K} s_{mrp} \geq 0, \ \forall r \in R \tag{3.54}$$

$$s_r + t_r - b_r \leq 1, \ \forall r \in R \tag{3.55}$$

$$s_r + t_r - 2 \cdot b_r \geq 0, \ \forall r \in R. \tag{3.56}$$

In the above, we identified the case where a register should be a BILBO or a CBILBO. If a register is used as a TPG and an SR in the *same* sub-test session p, the register should be reconfigured as a CBILBO. Otherwise, it should be a BILBO. First, let us check the case for a TPG. A binary variable t_{rp} is 1 if a register r is used as a TPG in a sub-test session p and is 0 otherwise. It is formulated as

$$\bar{t}_{rmlp} \cdot t_{rp} - \sum_{m \in M_i^c, l \in I(m)} t_{rmlp} \geq 0, \ \forall r \in R, p \in K. \tag{3.57}$$

Note that a binary variable t_{rmlp} is set to 1 only if a register r is the TPG for an input port l of a module m in a p sub-test session. Hence, Eq. (3.57) sets the binary variable t_{rp} to 1 if a register r is a TPG of *any* module in a sub-test session p.

Next, let us check the case for an SR. A binary variable s_{rp} is 1 if a register r is used as an SR in a sub-test session p and is 0 otherwise. The formulation is

$$\bar{s}_{mrp} \cdot s_{rp} - \sum_{m \in M_o^c} s_{mrp} \geq 0, \ \forall r \in R, p \in K. \tag{3.58}$$

Note that a binary variable s_{mrp} is 1 only if a register r is used as the SR for a module m in the p sub-test session. Hence, Eq. (3.58) sets the binary variable s_{rp} to 1 if a register r is used as an SR for *any* module in a sub-test session p.

Eq. (3.59) and (3.60) set a binary variable c_{rp} to 1 only if t_{rp}=1 and s_{rp}=1. In other words, c_{rp} is 1 only if a register r is reconfigured into a TPG and an SR in the same sub-test session p. Hence, the register should be reconfigured into a CBILBO. Finally, a binary variable c_r is set to 1 due to Eq. (3.61) if there is at least one sub-test session that requires a register r to be reconfigured as a CBILBO. This implies that the register r should be a BILBO if c_{rp} =0 (under b_r =1).

$$s_{rp} + t_{rp} - c_{rp} \leq 0, \quad \forall r \in R, p \in K \tag{3.59}$$

$$s_{rp} + t_{rp} - 2 \cdot c_{rp} \geq 0, \quad \forall r \in R, p \in K \tag{3.60}$$

$$\bar{c}_{rp} \cdot c_r - \sum_{p \in K} c_{rp} \geq 0, \ \forall r \in R. \tag{3.61}$$

3.4 Constants

Coefficients, such as in digital filters or signal processing applications, are usually constants. An operation in a DFG may receive constant values on an input port. If an input port of a module is connected to only one constant, it is usually hardwired to necessary constant logic values. The hardwired input port is resolved into the logic during the optimization process of a logic synthesis. So it is unnecessary to connect a TPG for the input port.

However, if an input port receives more than one constant, the input port is unlikely to be removed during the logic synthesis. Since the input port is not connected to any variable in the DFG, there is no register to be reconfigured into a TPG for the input port. To achieve a high fault coverage, it is necessary to add an extra TPG to the input port or to add an extra path to an existing TPG. Therefore, a case in which an input port is connected only to constants should be avoided in the assignment process if possible.

To avoid such a case, we introduce a new binary variable t^c_{cmlp}. The variable is 1 only if a constant c needs to be reconfigured into a TPG for an input port l of a module m in a p sub-test session. Thus, previous Eqs. (3.47) through (3.50) need to be modified as follows:

$$z_{cml} - \sum_{p \in K} t^{c}_{cmlp} \geq 0, \ \forall \ c \in C, m \in M_i^c, l \in I(m) \qquad (3.47')$$

$$\sum_{r \in R, p \in K} t_{rmlp} + \sum_{c \in C, p \in K} t^{c}_{cmlp} = 1, \forall\, m \in M_i^c, l \in I(m) \qquad (3.48')$$

$$\sum_{r \in R} t_{rm0p} + \sum_{c \in C} t^{c}_{cm0p} - \sum_{r \in R} t_{rm1p} - \sum_{c \in C} t^{c}_{cm1p} = 0, \ \forall\, m \in M_i^c, \ p \in K \qquad (3.49')$$

$$\sum_{r \in R} s_{mrp} - \sum_{r \in R} t_{rm0p} - \sum_{c \in C} t^{c}_{rm0p} = 0, \ \forall\, m \in M_i^c, \ p \in K. \qquad (3.50')$$

By assigning a high cost in setting t^{c}_{cmlp} to 1, this case can be avoided when possible.

3.5 Objective Function

The objective of an ILP is to minimize an objective function, i.e., cost function, for the given ILP formulation. The cost function for the proposed ILP model represents hardware area (in terms of transistor count) and is given below.

$$w_r\ N_r + w_s\ N_s + w_t\ N_t\ + w_b\ N_b + w_c\ N_c + w_i N_i + w_{t^c} N_{t^c}$$

where N_r, N_s, N_t, N_b, N_c, and N_i are the number of system registers, SRs, TPGs, BILBOs, CBILBOs, and interconnections, respectively. A variable N_{t^c} is the number of constants that should be reconfigured into TPGs. Weights, $w_r, w_s, w_t, w_b,$ *and* w_c are the number of transistors for the corresponding hardware. The weight w_i is the average number of transistors per input for various size multiplexers; it was obtained as the average of the number of transistors for an n-input multiplexer divided by n for n = 2, 3, …7. The weight w_{t^c} is the cost for constants. We assign w_{t^c} a number greater than any other weight to avoid, if possible, the case in which a port needs an extra TPG or an extra wire to an existing TPG.

To compute the value (cost) of the objective function, it is necessary to compute the number of individual registers, BIST registers, and multiplexers. They are readily calculated by summing up the related binary variables as shown below:

- Number of CBILBOs: $N_c = \sum_{r \in R} c_r$ (3.62)
- Number of BILBOs: $N_b = \sum_{r \in R} b_r - N_c$ (3.63)
- Number of SRs: $N_s = \sum_{r \in R} s_r - N_c - N_b$ (3.64)
- Number of TPGs: $N_t = \sum_{r \in R} t_r - N_c - N_b$ (3.65)
- Number of Registers: $N_r = N_R - (N_s + N_t + N_b + N_c)$ (3.66)

  where N_R represents the minimum number of registers necessary for a given DFG.
- Number of constants to be reconfigured as TPGs: $N_{t^c} = \sum_{c \in C, m \in M_i^c, l \in I(m), p \in K} t^c{}_{cmlp}$ (3.67)
- Number of interconnections: $N_i = \sum_{r \in R, m \in M_i^c, l \in I(m)} z_{rml} + \sum_{m \in M_o^c, r \in R} z_{mr}$ (3.68)

Note that the summation for deriving $N_c, N_b, N_s, N_t,$ and N_r is the same as Eq. (3.29) – Eq. (3.33). And the number of interconnections is derived instead of number of multiplexers in region-wise ILP based approach.