On ILP Formulations for Built-In Self-Testable Data Path Synthesis

Han Bin Kim
Dept. of Electr. & Comput. Eng.
Virginia Tech, Blacksburg
VA 24061-0111
1-540-231-4942
hanbink@ee.vt.edu

Dong Sam Ha
Dept. of Electr. & Comput. Eng.
Virginia Tech, Blacksburg
VA 24061-0111
1-540-231-4942
ha@vt.edu

Takeshi Takahashi
Advantest Lab. Ltd.
48-2 Matsubara, Kamiayashi, Aoba-ku
Sendai, Miyagi 989-31, Japan
81-22-392-8731
takeshi@atl.advantest.co.jp

ABSTRACT

In this paper, we present a new method to the built-in self-testable data path synthesis based on integer linear programming (ILP). Our method performs system register assignment, built-in self-test (BIST) register assignment, and interconnection assignment concurrently to yield optimal designs. Our experimental results show that our method successfully synthesizes BIST circuits for all six circuits experimented. All the BIST circuits are better in area overhead than those generated by existing high-level BIST synthesis methods.

Keywords

high-level BIST synthesis, built-in self-test, BIST, ILP.

1. INTRODUCTION

High-level built-in self-test (BIST) synthesis aims to embed BIST capability for the synthesized circuits, and it usually employs a BIST architecture called the parallel BIST [1]-[6]. The parallel BIST, which is based on random pattern testing, assigns a test pattern generator and a test data evaluator for every input/output port of a module (often combinational circuit) under test.

One of the earliest high-level BIST synthesis methods was proposed by Papachristou et al. [1]. For their method, all operations and variables are assigned to avoid self-adjacent registers, which are undesirable in BIST due to high area overhead. Their method was refined later to further reduce the area overhead [2]. Avra proposed an elegant solution to avoid self-adjacent registers based on register conflict graphs [3]. Parulkar et al. investigated a method that maximizes the sharing of test registers to reduce the area overhead [4]. The above mentioned methods focus on minimization of area overhead in BIST synthesis. To reduce test time in BIST, Harris and Orailoglu examined conditions that prevent concurrent testing of modules [5]. In our earlier work, we proposed a method that considers both area overhead and test time in the synthesis process [6]. Our method allocates signature registers first, so that the synthesized BIST circuit is guaranteed be tested in a *k*-test session where *k* is 1, 2, ...N and N is the number of modules.

Hafer and Parker pioneered formulating a high-level synthesis problem into an integer linear programming (ILP) model in the early 1980s [7]. Some recent works include Gebotys and Elmasry and Rim et al. [8] [9]. Gebotys and Elmasry applied an ILP to architectural synthesis where a scheduling and a module/register allocation were performed concurrently [8]. Rim et al. presented an ILP model to solve the binding problem focusing on minimizing hardware resources [9]. A major advantage of an ILP based approach is that the obtained solution is optimal though computationally intensive due to the inherent nature of ILP, which involves an exhaustive search.

In this paper, we propose an ILP based method that performs system register assignment, BIST register assignment, and interconnection assignment concurrently to achieve a global optimality. The objective for our method is to minimize the area for each *k*-test session. Hence, our BIST circuit is minimal in area overhead for each *k*-test session. Hence, like our earlier method presented in [6], our ILP-based method offers a range of designs with different figures of merit in area and test time.

We assume that readers understand basics of ILP. Otherwise, readers may refer to [10]. The paper is organized in the following manner. In Section 2, we briefly explain high-level synthesis and describe necessary terms. In Section 3, we describe the proposed ILP formulations for BIST register and interconnection assignments. Section 4 contains experimental results. Section 5 concludes the paper.

2. BACKGROUND

High-level synthesis involves scheduling, module assignment, and register assignment. Among the three operations, register assignment has the greatest impact on the parallel BIST and hence is the subject of the paper. *In this paper, we consider DFGs in which scheduling and module assignment have been completed. All examples given in this section are for the DFG and data path given in Fig. 1.*

Gray lines in the DFG denote clock cycle boundaries called control steps. A register should be assigned to each input or output variable on a clock boundary, and this process is called register assignment. If two variables overlap at a control step, the two variables are *incompatible*. Two incompatible variables should be assigned to two different registers. The data path in Fig. 1(b) is obtained under a register assignment: R0 = {0, 4}, R1 = {1, 3, 6}, and R2 = {2, 5, 7}.

The *horizontal crossing* of a control step is the number of variables for the control step. Therefore, the minimum number of registers required for the synthesis of a DFG is equal to the maximal horizontal crossing of the DFG. The minimum number of modules necessary for a type of operation is directly obtained from the maximum currency of the operation. The data path logic in Fig. 1(b) contains the minimum number of registers (three) and the minimum number of modules (two). *We assume that the numbers of registers and modules to be used for the synthesis of a DFG are known a priori.*

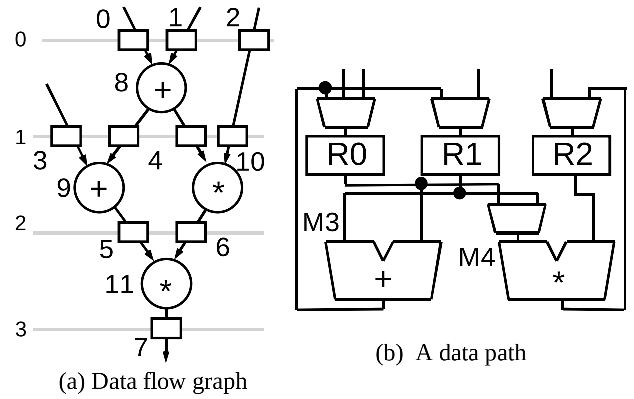

Figure 1. A data flow graph and a synthesized data path

2.1 Terms

The following nomenclatures are defined for a DFG and are used for our ILP model described in this paper.

- V^o is the set of operations: $V^o = \{8, 9, 10, 11\}$.
- V^v is the set of variables: $V^v = \{0, 1, 2, \ldots, 7\}$.
- l is the label of an input port of an operation. The leftmost input port is labeled as 0, the next right port as 1, and so on. An input port is designated by its label.
- *I(o)* is the set of input ports for an operation *o*: *I(8)* = {0,1} and *I(9)* = {0,1}.
- E^i is the set of ordered triples (*v*, *o*, *l*) defined for the *inputs* of all operations where *o* is an operation, *l* is an input port of the operation, and *v* is the variable on the input port *l:* E^i = {(0,8,0), (1,8,1), (3,9,0), (4,9,1), (4,10,0), (2,10,1), (5,11,0), (6,11,1)}.
- E^o is the set of ordered doubles (*o*, *v*) defined for the *outputs* of all operations where *o* is an operation and *v* is the output variable of the operation: E^o ={(8,4), ((9,5), (10,6), (11,7)}.
- *T* is the set of control steps: *T* = {0, 1, 2, 3}.
- *C* is the set of constants: *C* = ∅.

The following nomenclatures are defined for a data path logic to be synthesized from a DFG.

- *R* is the set of the registers: *R* = {0, 1, 2}.
- *M* is the set of modules: *M* = {3, 4}.
- *I(m)* is the set of input ports of a module m where m∈*M*: *I(3)* = {0,1} and *I(4)* = {0,1}.

A binary variable x_{vr} (x_{om}) is 1 only when a variable (operation) *v* (*o*) is assigned to a register (module) *r* (*m*) and is 0 otherwise. Each variable (operation) should be assigned to only one register (module).

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*. Similarly, a binary variable z_{mr} is 1 only if there is an interconnection between the output of a module *m* to a register *r* and is 0 otherwise. For details on register assignment and interconnect assignment, refer to [9] and [10].

2.2 BIST Synthesis and Test Registers

Two constraints are imposed in our BIST synthesis (and in most BIST synthesis systems). *First, test pattern generators and signature registers are reconfigured from existing system registers.* In other words, all test registers function as system registers during normal operation. *Second, extra paths are not added for testing.* It can be seen easily that the constraints can be met through reconfiguration of existing registers into four different types of test registers described below.

A system register may be converted into one of four different types of test registers: a test pattern generator (TPG), a multiple input signature register (short for signature register), a built-in logic block observer (BILBO), or a concurrent BILBO (CBILBO) [11],[12]. If a register should be a TPG and a signature register (SR) at the same sub-test session, it should be reconfigured as a CBILBO. If a register should behave as a TPG and an SR, but not at the same time, it should be reconfigured as a BILBO. Reconfiguration of a register into a CBILBO requires double the number of flip-flops of the register.

3. PROPOSED ILP FORMULATIONS

In the following, we present ILP formulations for BIST register assignments and interconnection assignments, multiplexer assignments, which are derived directly from interconnection assignments. 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, where *k* is 1, 2, ...N and N is the number of modules of the DFG. Hence, our method finds an optimal (in area) BIST design for each *k*-test session.

3.1 Interconnection Assignment

One of the constraints for the proposed BIST synthesis is not to add extra paths dedicated for testing. 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.

Using an auxiliary binary variable z_{vroml}, the condition can be formulated as follows for non-commutative modules.

$$\sum_{v\in V^v, o\in V^o} z_{vroml} - z_{rml} \ge 0,\ \forall\, r\in R, m\in M, l\in I(m) \qquad (1)$$

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

Upon z_{rml} =1, Eq. (1) requires at least one auxiliary binary variable z_{vroml} to be 1. From Eq. (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. In a similar manner, interconnections from modules to registers can be formulated.

Commutative operations, in which the two input ports can be swapped, are modeled as follows [9]. Inputs are applied to input ports of the operation through pseudo-input ports. Let a binary variable $s_{l^*,l,o} = 1$ if a connection exists between a pseudo-input port l^* and an input port l of an commutative operation o. Hence, Eq. (3) should replace Eq. (2) for a commutative operation.

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

3.2 Multiplexer Assignment

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 (4) and (5).

$$\sum_{m \in M} z_{mr} = m_r,\ \forall r \in R \quad (4)$$

$$\sum_{r \in R} z_{rml} = m_{ml},\ \forall m \in M, l \in I(m) \quad (5)$$

The total number of n-input multiplexers is obtained by scanning all m_{ml} variables for all inputs of registers and modules.

3.3 BIST Register Assignment

The proposed method tries to find an optimal 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 lables 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. We present ILP formulas for BIST register assignments in the following.

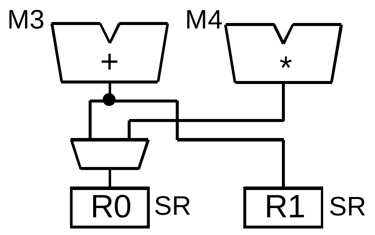

Figure 2. A partial data path for an SR assignment

3.3.1 Signature Register 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. 2.* Note that the partial data path, when registers are properly reconfigured for BIST, can be tested in one test session or two test sessions.

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.

$$z_{mr} - \sum_{p \in K} s_{mrp} \geq 0,\ \forall m \in M, r \in R \quad (6)$$

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) for K={1, 2}. 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,

$$\sum_{r \in R, p \in K} s_{mrp} = 1,\ \forall m \in M \quad (7)$$

Example: $s_{3,0,1} + s_{3,1,1} + s_{3,0,2} + s_{3,1,2} = 1$ and $s_{4,0,1} + s_{4,0,2} = 1$ for module M3 and module M4

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

$$\sum_{m \in M} s_{mrp} \leq 1,\ \forall r \in R, p \in K \quad (8)$$

For register R0 under the *two*-test session, Eq. (8) 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.

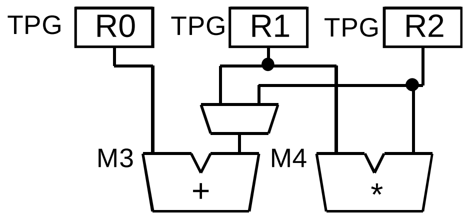

Figure 3. A partial data path for a TPG assignment

3.3.2 Test Pattern Generator Assignment

In this subsection, we formulate the assignment of TPGs. *All the illustrations given in this subsection refer to Fig. 3.* Note that the data path can be tested in a *1*-test session or a *2*-test session.

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.

$$z_{rml} - \sum_{p \in K} t_{rmlp} \geq 0,\ \forall r \in R,\ m \in M,\ l \in I(m) \quad (9)$$

$$\sum_{r \in R, p \in K} t_{rmlp} = 1, \forall m \in M,\ l \in I(m) \quad (10)$$

From (9), 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. (10)

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.

$$\sum_{r\in R} t_{rm0p} - \sum_{r\in R} t_{rm1p} = 0,\ \forall\, m\in M,\ p\in K \tag{11}$$

$$\sum_{r\in R} s_{mrp} - \sum_{r\in R} t_{rm0p} = 0,\ \forall\, m\in M,\ p\in K \tag{12}$$

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 (11), and $s_{3,0,1} + s_{3,1,1} - t_{0,3,0,1} = 0$ from (12).

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,

$$t_{rm0p} + t_{rm1p} \le 1,\ \forall\, r\in R,\ m\in M,\ p\in K \tag{13}$$

3.3.3 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.

$$\bar{x}_i \cdot x - \sum_i x_i \ge 0, \text{ where } \bar{x}_i \text{ is a constant equal to the number of } x_i\text{s.} \tag{14}$$

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 (14). 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 are1, then the register r should be reconfigured into a BILBO or a CBILBO. Eq. (15) determines if a register r is used as a TPG, and Eq. (16) 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 (17) and (18). Hence, b_r =1 indicates that the register r should be reconfigured into either a BILBO or a CBILBO.

$$\bar{t}_{rmlp} \cdot t_r - \sum_{m\in M, l\in I(m), p\in K} t_{rmlp} \ge 0,\ \forall r\in R \tag{15}$$

$$\bar{s}_{mrp} \cdot s_r - \sum_{m\in M, p\in K} s_{mrp} \ge 0,\ \forall\, r\in R \tag{16}$$

$$s_r + t_r - b_r \le 1,\ \forall\, r\in R \tag{17}$$

$$s_r + t_r - 2\cdot b_r \ge 0,\ \forall\, r\in R \tag{18}$$

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

$$\bar{t}_{rmlp} \cdot t_{rp} - \sum_{m\in M, l\in I(m)} t_{rmlp} \ge 0,\ \forall r\in R, p\in K. \tag{19}$$

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. (19) 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

$$\bar{s}_{mrp} \cdot s_{rp} - \sum_{m\in M} s_{mrp} \ge 0,\ \forall\, r\in R, p\in K. \tag{20}$$

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. (20) 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. (21) and (22) 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. (22) 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).

$$s_{rp} + t_{rp} - c_{rp} \le 1,\ \ \forall\, r\in R, p\in K \tag{21}$$

$$s_{rp} + t_{rp} - 2\cdot c_{rp} \ge 0,\ \ \forall\, r\in R, p\in K \tag{22}$$

$$\bar{c}_{rp} \cdot c_r - \sum_{p\in K} c_{rp} \ge 0,\ \forall\, r\in R \tag{23}$$

3.3.4 Constants

An operation in a DFG may receive constant values on an input port. If an 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. Therefore, a case in which an input port is connected to only constants should be avoided in the assignment process if possible. In order to avoid such as a case, previous equations (9) through (12) need to be modified slightly. To conserve space, the modified formulas are omitted.

3.4 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.

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

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. Necessary ILP formulations for computation of the numbers are not shown due to the space limit. 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.

3.5 Reduction of the Search Space

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. Since all the possible assignments are isomorphic, an arbitrary selection of an assignment does not affect the quality of the solution. However, it reduces the search space by *n*!. We employ the method for our ILP model to reduce the search space.

4. EXPERIMENTAL RESULTS

In this section, we present experimental results on the performance of our proposed method, called ADVBIST. We also compare the performance of ADVBIST with three other BIST synthesis methods: RALLOC proposed by Avra, BITS proposed by Parullkar et al., and our previous method ADVAN [3],[4],[6]. For the implementation of RALLOC and BITS, we followed the algorithms presented in [3] and [4], respectively. We used a commercial ILP solver to solve the ILP formulations of ADVBIST [13].

4.1 Background

We measured the performance of the four BIST synthesis systems for six data. Two data flows called tseng and Paulin, respectively, are widely adopted for benchmarking high-level BIST synthesis. The other four data flow graphs are a 6th order FIR (finite impulse response) filter, a 3rd order IIR (infinite impulse response) filter, a 4-point DCT (discrete cosine transformation) circuit, and a 6-tap wavelet filter. The other four data flow graphs were synthesized using HYPER [14]. The width of the data path logic is eight for all the circuits. The reference circuits, which were used to measure the area overhead of BIST designs, were obtained through an ILP for data path synthesis. The reference circuits are optimal in area.

In this paper, the area of a circuit is represented by the transistor count of registers and multiplexers in the circuit. *The data path logic of a circuit is not considered in the transistor count.* The number of transistors in test registers and multiplexers is based on the circuits of [11] and [12] and is given in Table 1. In the table, #Trs and #MuxIn denote the number of transistors and the number of multiplexer inputs, respectively. Note that the transistor counts of registers and multiplexers are the weights of our objective function described in Section 3.4.

Table 1. Number of transistors of 8-bit test registers and multiplexers

a) Test registers

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

b) Multiplexers

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

4.2 Experimental Results

We conducted experiments for the six circuits to measure the performance of the proposed method. We set a limit on the execution time, which is 24 CPU hours, in running the ILP solver. Hence a solution may not be optimal if obtained without searching all the possible solution space due to the time limit. Experimental results on area overhead (%) and processing time of ADVBIST for each test session are given in Table 2. The entries marked with "*" are the ones whose processing time reached the CPU time limit.

Table 2: Performance of the proposed method ADVBIST

Ckt		*k*=1	*k*=2	*k*=3	*k*=4
tseng	overhead	33.8	28.2	25.7	-
	time	58s	1m 22s	35s	-
paulin	overhead	37.5	28.1	25.3	25.3
	time	4h 42m 0s	24m 55s	11m 40s	59m 34s
fir6	overhead	30.1	21.2	15.3	-
	time	17m 34s	40m 16s	23h 56m 4s	-
iir3	overhead	23.6	17.3	16.3	-
	time	3h 11m 8s	2h 6m 26s	2h 50m 8s	-
dct4	overhead	23.3*	24.9*	45.5*	28.3*
	time	24h	24h	24h	24h
wavelet6	overhead	13.9	11.3	11.3	-
	time	11m 9s	10h 5m 15s	14h 39m 24s	-

As shown in the table, ADVBIST successfully synthesized a BIST data path for each test session for all six circuits. Among the 20 BIST designs obtained by ADVBIST, 16 circuits are optimal (i.e., minimal) in area, and the four circuits of dct4 may not be optimal.

The area overhead of ADVBIST ranges from 11 percent to 46 percent. Since the area overhead of a circuit is calculated without considering the area for the data path logic modules, the actual area overhead will be much lower than the ones presented in the table. For example, the area overhead of paulin is 37.5 percent for *k*=1. The actual area overhead would be reduced to about 15.9 percent under a certain implementation of the data path logic. Therefore, the area overhead of ADVBIST is small to moderate for all six circuits. Note that the area overhead of the 16 optimal BIST designs is minimal and cannot be reduced any further.

We compare the performance of ADVBIST with three other BIST synthesis systems, ADVAN, RALLOC and BITS [3],[4],[6]. To make the comparison meaningful, the six data flow graphs used in the experiment employed the same scheduling and the same module assignment for all four BIST systems. The performance of the four high-level BIST synthesis systems is presented in Table 3. The table shows the case in which the number of test sessions is maximal for a given circuit. The number of test sessions for a circuit is given under the circuit name in the table. Column headings for the table are explained below.

R:	total number of registers,
T (S):	number of TPGs (SRs),
B (C):	number of BILBOs (CBILBOs),
M:	total number of multiplexer inputs,
Area:	total number transistors of the registers and the multiplexers,
OH:	the area overhead of the BIST design (%)

From the table, ADVBIST performs better than the other systems in area overhead for all circuits. Since five of the six BIST circuits of ADVBIST are optimal in area, they are expected to be better than other designs. However, even the non-optimal BIST circuit for dct4 is better than other designs. For some circuits, area overhead of ADVBIST is substantially less than that for the other methods. For example, the area overhead of ADVBIST is 11 percent for wavelet6, while that for the other methods are in the range of 27 to 45 percent. It should be pointed out that ADVBIST and our previous method ADVAN do not add any additional registers. However, RALLOC needs one additional register for fir6, iir3, and wavelet6, while BITS requires one additional register for dct4. The addition of registers incurs large area overhead as can be seen in Table 4.

Table 3. Performance of various high level BIST synthesis systems

Ckt	Method	R	T	S	B	C	M	Area	OH(%)
tseng (3)	Ref.	5					14	1600	
	ADVBIST	5	2	1	2	0	14	2152	25.7
	ADVAN	5	2	1	0	0	23	2368	32.4
	RALLOC	5	1	0	3	0	14	2300	30.4
	BITS	5	2	1	1	0	20	2436	34.3
paulin (4)	Ref.	5					19	1856	
	ADVBIST	5	2	2	1	0	23	2484	25.3
	ADVAN	5	3	1	0	0	26	2684	30.8
	RALLOC	5	1	0	3	0	25	2892	35.8
	BITS	5	2	0	0	1	27	3024	38.6
fir6 (3)	Ref.	7					20	2576	
	ADVBIST	7	4	1	0	0	26	3040	15.3
	ADVAN	7	2	1	0	0	28	3308	22.1
	RALLOC	8	1	1	2	0	36	4212	38.8
	BITS	7	1	0	0	1	24	3280	21.5
iir3 (3)	Ref.	6					22	2224	
	ADVBIST	6	5	1	0	0	23	2656	16.3
	ADVAN	6	3	1	0	0	32	3432	35.2
	RALLOC	7	1	0	2	0	38	4212	47.2
	BITS	6	2	0	2	0	29	3176	30.0
dct4 (4)	Ref.	6					24	2320	
	ADVBIST	6	3	1	1	0	32	3236	28.3
	ADVAN	6	3	1	0	0	35	3420	32.2
	RALLOC	6	1	1	2	0	37	3812	39.1
	BITS	7	1	1	0	1	38	4180	44.5
wavelet6 (3)	Ref.	7					25	2880	
	ADVBIST	7	2	2	0	0	31	3248	11.3
	ADVAN	7	2	1	0	0	46	4182	31.1
	RALLOC	8	1	0	3	0	50	5186	44.5
	BITS	7	1	0	2	0	40	3946	27.0

Close examination of the BIST circuits reveals that the better performance of ADVBIST is largely due to less multiplexer area. The number of multiplexers and the size of individual multiplexers are sensitive to the BIST register assignment, which differentiates essentially the four BIST synthesis systems.

5. SUMMARY

In this paper, we presented a new approach based on integer linear programming (ILP) that performs the three register assignment tasks concurrently to yield optimal designs. In addition, our approach finds an optimal register assignment for each *k*-test session. Our experimental results show that the proposed method successfully synthesized BIST circuits for every *k*-test session for all six circuits experimented. Among 20 BIST circuits, 16 circuits are minimal in area overhead. Our method performs better for all circuits in area overhead than the other three BIST synthesis systems.

The major limitation of the proposed method lies in the long processing time for large designs. Several measures such as heuristics to reduce the search space and partition of designs may be employed to address the problem. Further research is necessary in this area to fully utilize the potential of ILP based approaches in high-level BIST synthesis.

6. REFERENCES

[1] C.A. Papachristou, S. Chiu, and H. Harmanani, "A data path synthesis method for self-testable designs, " *Proc. 28th Design Automation Conf.*, pp. 378-384, June 1991.

[2] H. Harmanani and C.A. Papachristou, "An improved method for RTL synthesis with testability tradeoff," *Intl. Conf. on Computer-Aided Design*, pp. 30-35, Nov. 1993.

[3] L.J. Avra, "Allocation and assignment in high-level synthesis for self-testable data paths," *Proc. Int. Test Conf.*, pp. 463-472, Oct. 1991.

[4] I. Parulkar, S. Gupta, and M.A. Breuer, “Data path allocation for synthesizing RTL designs with low BIST area overhead,” *Proc. 32nd Design Automation Conf.*, pp. 395-401, June 1995.

[5] A. Orailoglu and I.G. Harris, “Microarchitectural synthesis for rapid BIST testing,” *IEEE Trans. Computer-Aided Design*, Vol.16, No. 6, pp. 573-586, June 1997.

[6] H.B. Kim, T. Takahashi, and D.S. Ha, "Test session oriented built-in self-testable data path synthesis,” *Proc. Int. Test Conf.*, pp. 154-163, Oct. 1998.

[7] L. Hafer and A. Parker, “A formal method for the specification, analysis, and design of register-transfer level digital logic,” *IEEE Trans. on Computer-Aided Design*, Vol. 2, pp. 4-18, Jan. 1983.

[8] C.H. Gebotys and M.I. Elmasry, “Global optimization approach for architecture synthesis,” *IEEE Trans. Computer-Aided Design*, Vol. CAD-12, pp.1266-1278, Sept. 1993.

[9] M. Rim, A. Mujumdar, R. Jain, and R. DeLeone, “Optimal and heuristic algorithms for solving the binding problem,” *IEEE Trans. on VLSI Systems*, Vol. 2, No. 2, pp. 211-225, June 1994.

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

[11] Koenemann, B.J. Mucha, and G. Zwiehoff, “Built-in logic block observation techniques,” Proc. *Int. Test Conf.*, pp. 37-41, Oct. 1979.

[12] L.-T. Wang and E.J. McCluskey, “Concurrent built-in logic block observer (CBILBO),” *Proc. Int. Symp. on Circuits and Systems*, pp. 1054-1057, May 1986.

[13] *CPLEX* 6.0 *Reference Manual*, ILOG, 1998.

[14] M. Potkonjak and J. Rabaey, “A scheduling and resource allocation algorithm for hierarchical signal flow graphs,” *Proc. 36th Design Automation Conf.*, pp. 7-12, June 1989.