VLSI DESIGN
1998, Vol. 7, No. 2, pp. 211–224
Reprints available directly from the publisher
Photocopying permitted by license only

Timing-Driven Circuit Implementation

DIMITRIOS KARAYIANNIS[a] and SPYROS TRAGOUDAS[b,*]

[a]*Viewlogic Systems Inc., Fremont, CA 94538-6530;*
[b]*Computer Science Department, 2130 Faner Hall, Southern Illinois University at Carbondale, Carbondale, IL 62901*

(Received 29 February 1996)

We consider the problem of selecting the proper *implementation* of each circuit *module* from a cell library to minimize the propagation delay along every path from any *primary input* to any *primary output* subject to an upper bound on the total area of the circuit. Different module implementations may have different areas and delays on the paths. We show that the latter problem is NP-hard even for directed acyclic graphs with two implementations per module and no restrictions on the overall area of the circuit. We present a novel retiming based heuristic for determining the minimum clock period on *sequential circuits*. Although our heuristics may handle a bound on the total area of the circuit, emphasis is given on the timing issue.

Keywords: Circuit implementation, technology mapping, delay minimization, NP-complete algorithms

1. INTRODUCTION

The circuit implementation problem studied here is related to the technology mapping problem studied earlier by Brayton *et al.* [1], Keutzer [7], Pedram *et al.* [13], Touati *et al.* [16, 17], Chaudhary *et al.* [3] and others. The authors above examine the problem of mapping a Boolean network using gates from a finite size cell library. However, in this paper, we consider Boolean networks that have already been mapped.

We examine the problem of selecting, from a cell library, an *implementation* for each module so that the propagation delay along any path from any *primary input* to any *primary output* is minimum. It is desirable that the total area of the circuit does not exceed a given bound. Every module implementation may have different delays along the paths connecting different pairs of input-output (I/O) terminals. Different implementations may also have different areas. In [2, 11], a similar problem was studied, the *basic circuit implementation* problem (BCI), where different implementations may have different areas, but the delays within each module are uniform.[1]

The circuit implementation problem is very complex since many factors, i.e., delay, area, power, must be taken into consideration. How-

*Corresponding author.

[1]Note that the authors in [11] insist that there is a path between every pair of input-output terminals. In this paper we relax the latter constraint which does not necessary hold on mapped macro-based circuits where a module represents a macro in a specific technology.

ever, a recent trend in Computer Aided Design (CAD) focuses on timing driven applications, where priority is given to the maximum delay between any I/O path in a designed *combinational circuit*, or between any two flip-flops in a *synchronous sequential circuit* [15]. Thus, our primary goal is to select module implementations so that we minimize the maximum delay of a given circuit. For the sake of simplicity, we focus on the Timing-Driven General Circuit Implementation (TDGCI) model, where the module implementations are selected without considering the area of each implementation. However, the heuristic solutions presented in this paper can be easily extended to consider a bound on the total area of the implemented circuit.

We consider the pin-dependent MIS library delay model as was formulated in [3], where the arrival time *arrival*() at the output g_o of some module g is a complex function expressed as *arrival* $(g_o, C_{g_o}) = \max_{g_i \in \text{inputs}(g)}(r_{g_i,g_o} + R_{g_i,g_o} C_{g_o} + \textit{arrival}(g_i, C_{g_i}))$, where r_{g_i,g_o} is the *intrinsic* gate delay from input g_i to output g_o of g, R_{g_i,g_o} is the *drive* resistance of g corresponding to a signal transition at input g_i, C_{g_o} in the *load capacitance* seen at g_o, and *arrival* (g_i, C_{g_i}) is the arrival time at input g_i corresponding to load C_{g_i} seen at that input [3]. The load capacitance C_g depends on the input pin capacitances of the gates it is driving[2]. Observe that if all pin capacitances of all module implementations are the same we result to a more simplified delay model, the *simplified TDGCI* problem where every module implementation has local delays on its various paths that do not depend on the delays on paths of other modules in the circuit. That way, the delay along a path can be computed by simply adding the delays on the module edges on the path.

Most of the previous work in the literature is on a simpler model, the BCI problem [2, 11, 10]. In this model the delay at each module ouput does not depend on module paths. For example, the rising transition of an m-input CMOS *N AND* gate can be approximated as [18]: $t_r = R_p/n(mnC_d + C_r + kC_g)$, where R_p is the effective resistance of p-device in a minimum-sized inverter, n is the width multiplier for p-devices in this gate, k is the fan-out, m is the fan-in, C_g is the gate capacitance of a minimum-sized inverter, C_d is the source/drain capacitance of a minimum-sized inverter, and C_r is the routing capacitance [18]. A similar approximation is given for the falling transition. Observe that due to the different capacitances, the delay of one module may depend on the delay of a neighboring module. However, the authors in [2, 11, 10] have considered a *simplified BCI* model so that the delay along a path can be computed by summing the delays on the modules on the path.

Chan [2] has shown that the simplified BCI problem is NP-hard for circuits with tree topology. Furthermore, for the simplified BCI model, Chan [2] has given a pseudo-polynomial time algorithm for trees, and a heuristic for basic circuits modeled by directed acyclic grphs (dags). Later, Li *et al.* [11] showed that the simplified BCI problem in NP-hard. They also developed a pseudo-polynomial time algorithm that obtains optimal solutions for basic series-parallel circuits [11]. In addition, they proposed six heuristics for basic combinational circuits under the simplified BCI model, without actually providing that the BCI problem on combinational circuits in NP-hard in the strong sense [11]. We have shown that the latter problem is indeed NP-hard by reducing from the One-In-Three 3SAT problem. The reduction is given in the Appendix. The later was also recently, and independently, shown in [10] by reducing from the 3SAT problem. Both reductions hold for the restricted case of two implementations per module.

It appears that it is often the case in VLSI where parameters related to capacitances are ignored so that algorithmic solutions are obtained easier. This is for example the case in the clustering problem for delay minimization studied in [8, 12, 14], among others.

[2]The latter recursive formula is slightly different from the one in [3] since we only consider Boolean Networks that have already been mapped.

In another context, the authors in [9] allow similar simplifications when they perform a retiming in a sequential circuit. Retiming is a technique that allows repositioning of the existing flip-flops of a sequential circuit so that its operation is not modified and the clock-period is minimized. This is equivalent to maintaining the same number of flip-flops at each cycle. Leiserson and Saxe have presented efficient retiming algorithms [9]. In fact, in this paper we modify one of the algorithms in [9] to solve the TDGCI problem efficiently for sequential circuits. Thus, the heuristics proposed in this paper are derived by initially considering the simplified TDGCI problem. At this point we decide the implementation for every module. Subsequently, we report the actual delay according to the pin-dependent MIS library delay model. That way we derive accurate solutions in a more efficient (in terms of time response) manner.

The paper is organized as follows. In Section 2, we solve an interesting open problem and we show that the TDGCI problem remains NP-hard on directed acyclic graphs (combinational circuits), where all the modules in the library have the same pin capacitanes and *there are only two possible implementations for each module*. This is an improvement over the result in Li *et al.* [11].

In Section 3, we consider the TDGCI problem on general circuits and we given heuristics for both combinational and sequential circuits. In sequential circuits the goal is to minimize the clock period. We define the latter problem as that of selecting an implementation for each of the circuit modules, so that the clock-period of the circuit is minimized and the circuit function is preserved. As in [9], we assume delays on the combinational components only, and not on the flip-flops. In the first part of Section 3 we present a method for combinational circuits. This approach resembles the iterative improvement methodology, a popular approach for CAD, which needs to be modified so that it is more time efficient. We emphasize that iterative improvement turns out to be a very expensive (time consuming) operation, especially when one insists on working directly on the pin-dependent MIS library delay model, whereas our proposed method is much more efficient both in time response and surprisingly in quality of results when compared to iterative improvement. Then we present an $O(|V|^2|E|\log|V|)$ time approach that uses retiming techniques as well as the latter method to obtain efficient solutions for the circuit implementation problem on sequential circuits. Our proposed heuristics are then compared with two alternative approaches in Section 4.

Below we present some notation. A circuit is an interconnected set of modules. Each module has a number of input and output *pins*. A module with p input pins and q output pins is called a (p, q)-*module*. Some of the input pins of a (p, q)-*module* may be primary inputs and some output pins may be primary outputs.

2. THE COMPLEXITY OF THE TDGCI PROBLEM

We first show that the TDGCI decision problem is NP-complete on directed acyclic graphs with two implementations per module and uniform pin capacitances.

TIMING-DRIVEN CIRCUIT IMPLEMENTATION PROBLEM (TDGGI)
Input: A combinational logic circuit, two implementations for each module of the circuit, and a delay D.
Output: Is there an implementation, such that they delay of the given circuit is at most D?

We reduce from NP-complete problem in [5]:

MAXIMUM 2-SATISFIABILITY (MAX-2SAT)
Input: Set U of variables, collection C of clauses over U such that each clause $c_k \in C$, $1 \leq k \leq |C|$ has $|c_k| = 2$, positive integer $K \leq |C|$.
Question: Is there a truth assignment for U that simultaneously satisfies at least K of the clauses in C?

THEOREM 2.1 *The TDGCI decision problem is NP-complete in the strong sense even if the*

maximum number of possible implementations for each module is two.

Proof The TDGGI decision problem is easily shown to be in NP. Next, we trasform MAX-2SAT to TDGCI. Let $U=\{u_1, u_2,\ldots,u_n\}$ be a set of variables and $C=\{c_1, c_2,\ldots,c_m\}$ be a set of clauses making up an instance of MAX-2SAT. We shall construct an instance of TDGCI such that the MAX-2SAT instance is satisfiable if an only if the constructed TDGCI instance has an implementation with delay at most D. We obtain the TDGCI instance as follows.

Let l_1^k, l_2^k be the first and the second literals of clause c_k, $1 \leq k \leq |C|$, respectively. Note that any literal l_i^k is either equal to some u_j or its complement u_j', where $u_j \in U$. For every clause c_k, $1 \leq k \leq |C|$, we costruct two modules called variable modules and a module called clause module. We call these three modules the k^{th} block. The structure of the variables and clause modules is given below.

Each variable module is a (4, 4)-module and corresponds to a literal in the particular clause. For example, if $l_1^k = u_2'$ and $l_2^k = u_3$, we construct two variable modules which we label U_2^k and U_3^k, respectively. The structure of module U_2^k and U_3^k, respectively. The structure of module U_i^k is independent of whether the respective literal is u_i or u_i' and is given in Figure 1a. Let the r^{th} input and r^{th} output of module U_i^k be labeled U_i^{kr} and U_i^{kr}, respectively. The labeling of the module's inputs and outputs is also given in Figure 1a. The clause module C_k, corresponding to clause c_k is a (2, 2)-module. Let the r^{th} input and r^{th} output of module C_k be labeled C_k^r and C_k^r, respectively. The structure of clause module C_k is given in Figure 1b. Observe that it contains 2 internal nodes (this constraint can be removed but it simplifies the description of the reduction), and 5 internal edges, labeled e_i^k, $1 \leq i \leq 5$, respectively.

The variable and clause modules are connected as follows: If $l_i^k = u_i$, $u_i \in U$, then $\mathcal{U}_i^{k1}$ is connected to C_k^1. If $l_1^k = u_i'$, $u_i' \in U$, then $\mathcal{U}_i^{k2}$ is connected to C_k^1. Similarly, if $l_2^k = u_j$, $u_j \in U$, then $\mathcal{U}_j^{k1}$ is

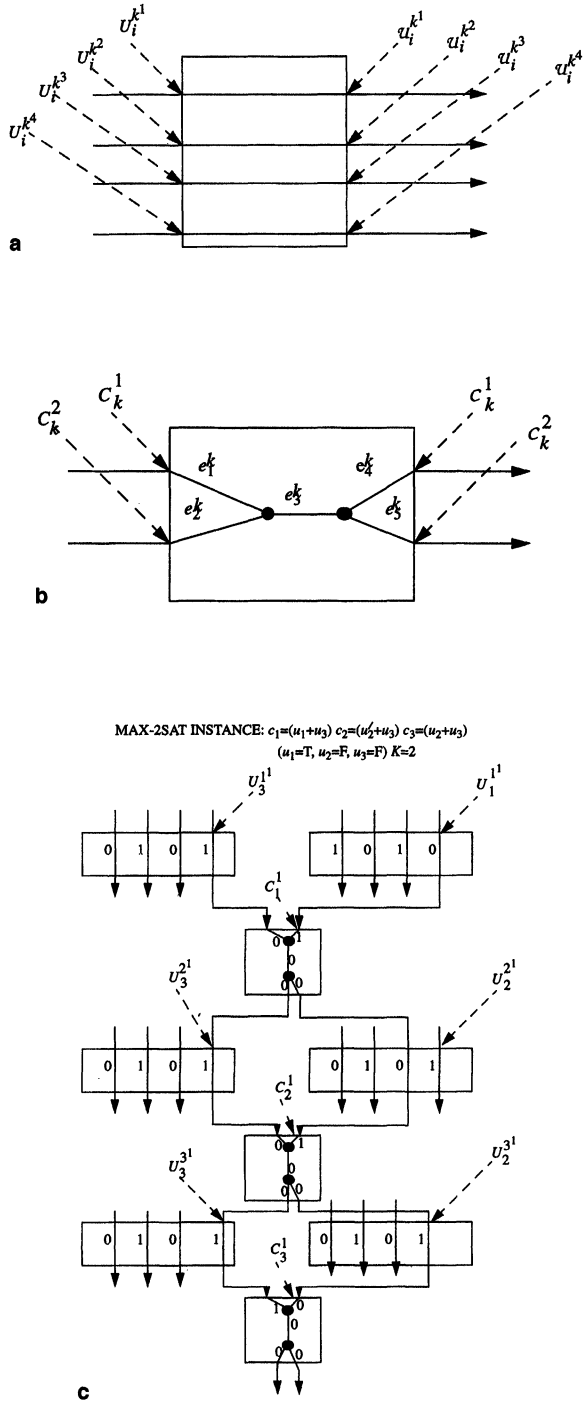

FIGURE 1 The reduction for the TDGCI problem. (a) Variable module U_i^k. (b) Clause module C_k. (c) The TDGCI instance (without enforcing consistency in the assignment of the true/false values). (d) Control module CU_i. (e) The complete TDGCI instance.

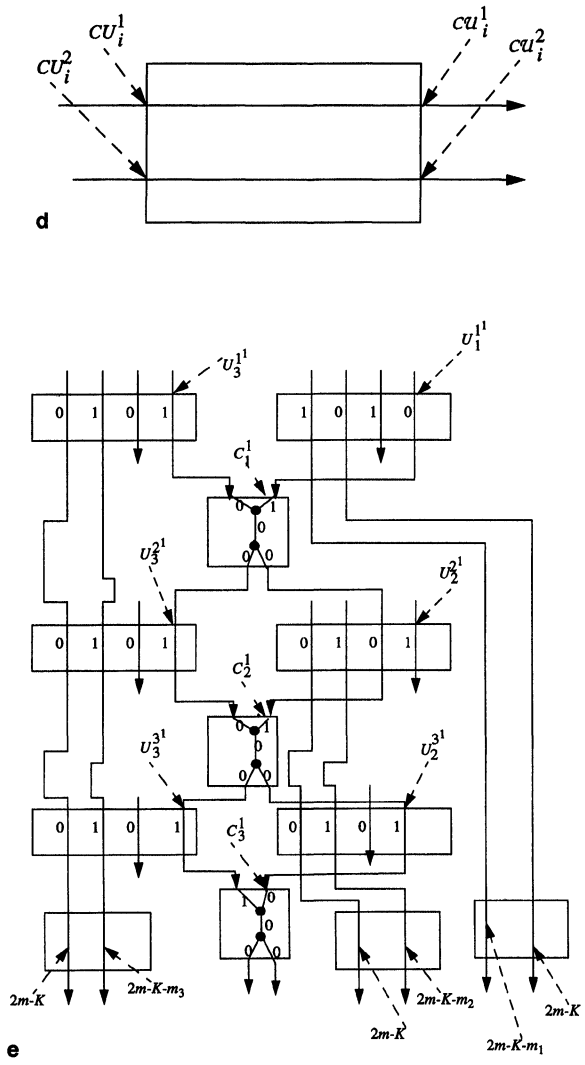

FIGURE 1 (Continued).

connected to C_k^2, and if $l_2^k = u'_j$, $u'_j \in U$, then U_j^{k2} is connected to C_k^2. Let $k' = k+1$. Furthermore, if $l_1^{k'}$ is u_m, $u_m \in U$, then C_k^1 is connected to $U^{k'1}_m$. If $l_m^{k'}$ is u'_m, $u'_m \in U$, then C_k^1 is connected to $U_m^{k'2}$. Similarly, if $l_2^{k'}$ is u_n, $u_n \in U$, then C_k^2 is connected to $U_n^{k'1}$, and if $l_2^{k'}$ is u'_n, $u'_n \in U$, then C_k^2 is connected to $U_n^{k'2}$.

Observe that up to this point we have specified only the connections on the first two inputs and outputs of every variable module in our design as well as all the interconnections to and from any clause module. (We postpone the remaining interconnections for later.)

We will now give two possible implementations for each variable and clause module, and an upper bound D on the delay, so that we guarantee that at most K clauses are satisfied if and only if the delay is at most D. *This will be shown assuming a consistent assignment of true/false values on all appearance of each variable. This consistency will be guaranteed later via connections of the 3rd and 4th inputs and outputs of each variable module.* Every variable module U_i^k has two implementations: If u_i is assigned the value true, then we assign delays 0, 1, 0, 1 along the edges connecting the i^{th} input and output, $1 \le i \le 4$, respectively. If u_i is assigned the value false, then we assign delys 1, 0, 1, 0 respectively. Similarly, each clause module C_k has two implementations. The implementation of a clause module depends on the true/false evaluation of the first literal of the respective clause. If the first literal of a clause is true then we choose the implementation where $e_1^k = 1$, and $e_2^k = e_3^k = e_4^k = e_5^k = 0$. Otherwise, we choose the second implementation where $e_2^k = 1$, and $e_1^k = e_3^k = e_4^k = e_5^k = 0$. Finally, we set the delay D to be $2m-K$. An example of the TDGCI instance corresponding to a MAX-2SAT instance is shown in Figure 1c.

We now show that the MAX-2SAT instance is satisfiable if and only if the constructed TDGCI instance has an implementation with delay D' at most D. Let c_k be a clause and consider the path from any of the first two inputs of the corresponding variable modules to any of the two outputs of the clause module c_k. If either one literal or both literals of c_k are evaluated to be true, then both outputs of C_k will have delay 1. If both literals are evaluated to be false, then both outputs will have delay 2. From the way we constructed the TDGCI instance, the delay up to the i^{th} block is added to the delay up to the $i+1^{\text{st}}$ block. Thus, we ensure that every satisfied clause c_k increments the delay up to the k^{th} block. On the other hand, if c_k is an nonsatisfied clause, the delay up to the k^{th} block is increased by 2 units.

If the MAX-2SAT instance is satisfied, there at least K clauses satisfied and $m-K$ clauses that are not satisfied. If there are exactly K clauses satisfied, then $D'=2(m-K)+K=2m-K$. It is easy to see that in the constructed TDGCI instance of Figure 1c, $D' \leq 2m-K$.

Conversely, suppose that the constructed TDGCI instance is satisfied. This implies that the delay along the longest path is at most $2m-K$, or equivalently, there are at least K blocks with delay 1. The latter implies that there are at least K clauses that are satisfied in the MAX-2SAT instance.

Up to now, we have assumed that we can ensure a consistant true/false assignment on all appearances of each variable. This can not be ensured necessarily with the up to now construction. However, we will enforce the latter by enlarging our construction as follows: For every variable u_i of the MAX-2SAT instance, we construct a (2, 2)-module which we call control module and we label it as CU_i. Let the r^{th} input and r^{th} output of Module CU^i be labeled CU_i^r and $C\mathcal{U}_i^r$, respectively. The structure of such a module is given in Figure 1d. Note, that a variable u_i may appear in more than one clauses, and for each of its appearances we have created a new variable module. We connect these variable modules with their coresponding control modules as follows: We connect the 3^{rd} and 4^{th} outputs of the variable module corresponding to the k^{th} appearance of variable u_i, $1 \leq k$, with the 3^{rd} and 4^{th} inputs, respectively, of the variable module corresponding to the next $(k+1^{\text{st}})$ appearance of variable u_{ik}. We continue in this manner and last we connect the 3^{rd} and 4^{th} outputs of the variable module corresponding to the last appearance of variable u_i with the 1^{st} and 2^{nd} inputs, respectively, of the control module CU_i. Let m_i be the number of clauses that contain either u_i or u_i'. Every control module CU_i has two implementations: If u_i is assigned the value true, then we choose the 1^{st} implementation where the edge connecting CU_i^1 and $C\mathcal{U}_i^1$ has delay $2m-K$, and the edge connecting CU_i^2 and $C\mathcal{U}_i^2$ has delay $2m-K-m_i$, respectively. We call these edges the 1^{st} and 2^{nd} internal edge of CU_i, respectively. If u_i is assigned the value false, then we choose the 2^{nd} implementation where the latter edges have delays $2m-K-m_i$, $2m-K$, respectively. Figure 1e shows the complete construction of the reduction of Figure 1c, and illustrates the latter construction.

We now show that the control modules guarantee a consistent true/false assignment to the variables. If the delay in the TDGCI instane is at most $2m-K$, then the delays up to $C\mathcal{U}_i^1$, $C\mathcal{U}_i^2$ of every control module CU_i must be at most $2m-K$. However, our construction enforces that exactly one of the two internal edges of CU_i is assigned delay of exactly $2m-K$ units. If the first internal edge of CU_j is assigned delay $2m-K$, then the delay up to input CU_j^1 is 0. This in turn implies that the delays on all edges connecting the 3^{rd} input and output of every variable module corresponding to variable u_i must be 0. The latter enforces a consistent true assignment to all appearances of variable u_j. Similarly, if the second internal edge of module CU_j is assigned delay $2m-K$, then it means that we ensure a consistent false assignment to all appearances of variable u_j.

3. HEURISTICS FOR THE TDGCI PROBLEM

3.1. Combinational Circuits

We propose a heuristic for solving the TDGCI problem on combinational circuits modeled by dags. For simplicity, we assume two implementations per module. We borrow ideas from the *iterative improvement* methodology. We first define the gain $g(u)$ of a node u to be the difference between the delay of the longest path considering the node's current and complimentary implementation, respectively. Let the two delays be denoted D_1 and D_2. The gain of node u is $g(u)=D_1-D_2$ i.e., the benefit resulting from the interchange on the module's implementation. In order to speed up operations however, we *employ a technique that*

calculates the gain of the modules approximately (based on local principles), in most cases provably correct. However, in order to be able to perform these local computations we must assume that all modules in the library have identical pin capacitances and therefore the load capacitance seen at each module output is the same. Note that this is where our approach is different from the straightforward iterative improvement. The major operation of our heuristic is to select the module with the *biggest calculated gain*, change its implementation has been found, and (b) it will not be changed in later iterations of the program[3]. Finally, after all circuit modules have been assigned an implementation we perform an extra step (topological search) to enhance the quality of our results. We consider the modules actual pin capacitances and via a longest path computation we calculate precisely the (minimized) maximum delay of the circuit.

In a preprocessing step, we transform the input combinational circuit to an acyclic directed graph, $G=(V, E)$, by substituting every (p, q)-module by $(p+q)$ vertices. Figure 2 illustrates the transformation. Note that in the transformed graph, delays exist only on the edges that resemble edges internal to the modules in the original circuit.

Next we describe the procedure that calculates the gains. Each vertex u contains two fields *lp-in* 1(u) and *lp-in2*(u). The former stores the maximum delay onthe longest path from S to u, and the latter the maximum delay on the longest path from D to vertex u. We calculate the value of both fields via two longest path computations, one from S to D, and one from D to S.

Let M_1 be some (p, q)-module, selected for evaluation. The heuristic adds lp_in1 (p_i) to the lp_in2 (q_j), for every pair (p_i, q_j) of M_1. It then adds the result to the weight of the internal edge from p_i to q_j. Observe that this is the maximum delay on the longest path from S to D passing through vertices p_i and q_j. Let's call this maximum delay $l^1_{\max}$. Next, it compares the value of $l^1_{\max}$ with the

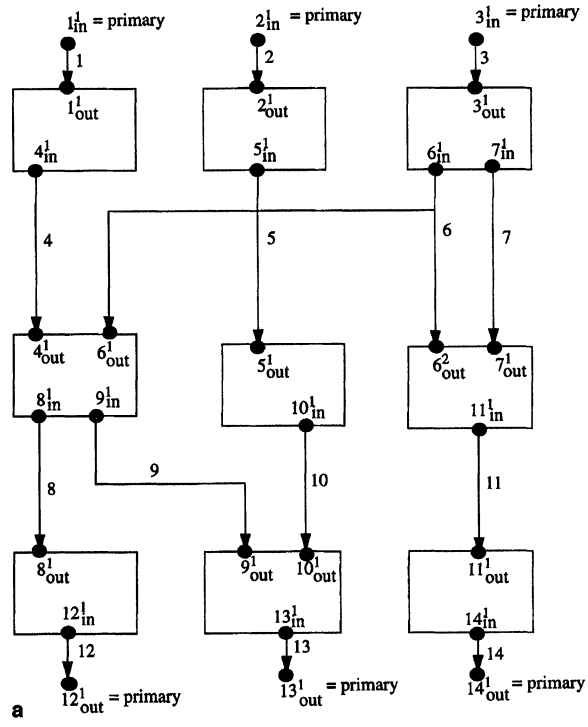

FIGURE 2 Constructing graph G for combinational circuits. (a) A combinational circuit. (b) The graph that corresponds to the circuit C of (a). Each net of C is a set of nodes (a node per pin) connected with external edges. For each module of C graph G contains a set of internal edges (presented inside the big cycles). Delays (>0) exist only on internal edges. These delays correspond to the internal delays of the modules in C.

value of the maximum delay on the global longest path from S to D, call it $l_{\max}$. If $l_{\max} - l^1_{\max} = 0$, then vertices p_i, q_j are on the global longest path from S to D. If $l_{\max} - l^1_{\max} > 0$, then vertices p_i, q_j are not on the global longest path from S to D. Either way, a flag is set to determine the status of each vertex. Let $l^2_{\max} = lp\_in1(p_i) + lp\_in2(q_j) + d_2$, where d_2 is the delay of the second implementation of M_1 from p_i to q_j.

The heuristic completes $g(M_1)$ as follows: If $l^2_{\max} > l_{\max}$, then $g(M_1) = l_{\max} - l^2_{\max}$. If $l^2_{\max} < l_{\max}$ and vertices p_i, q_i are not on global longest path, then $g(M_1)=0$. If $l^2_{\max} = l_{\max}$, then $g(M_1)=0$. If $l^2_{\max} < l_{\max}$ and vertices p_i, q_j are on

[3]This process is in fact repeated a fixed number of times of unlocking all cells, in hope of further reducing the overall delay.

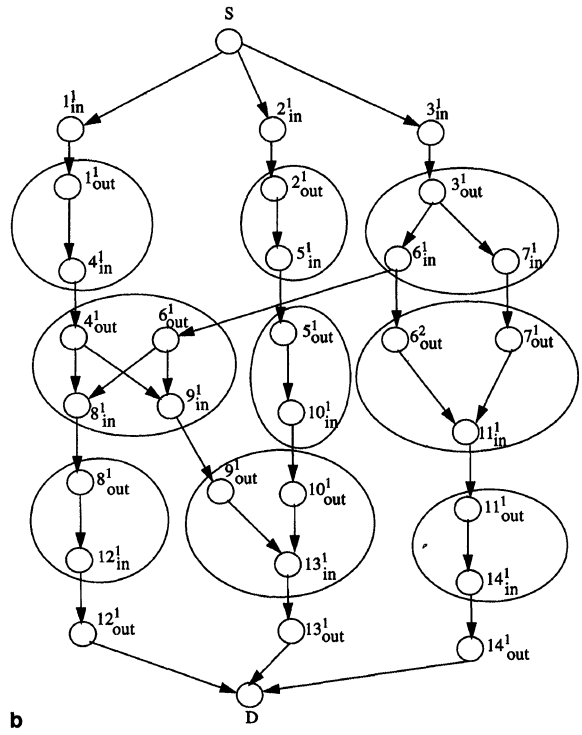

FIGURE 2 (Continued).

the global longest path, then $g(M_1) = l_{max} - l^2_{max}$. For every pair (p_i, q_j) of M_1 there is a corresponding gain for M_1. The smallest of these gains is the one that our heuristic considers as $g(M_1)$. Let $|V|$ be the number of modules of the given circuit, and $|E|$ to the number of all edges of the graph. The heuristic requires $O(|V||E|)$ time. This is much less than the straightforward iterative improvement scenario (even for the simplified TDGCI model) where every time a module changes implementation we have to compute the gain exactly.

3.2. Sequential Circuits

A straightforward heuristic that obtains optimal solution to the TDGCI problem on sequential circuits is based on the idea of generating a combinational circuit by selecting flip-flops to break the cycles, and then apply the algorithm above. Figure 3 illustrates the process. Our proposed approach uses the concept of retiming.

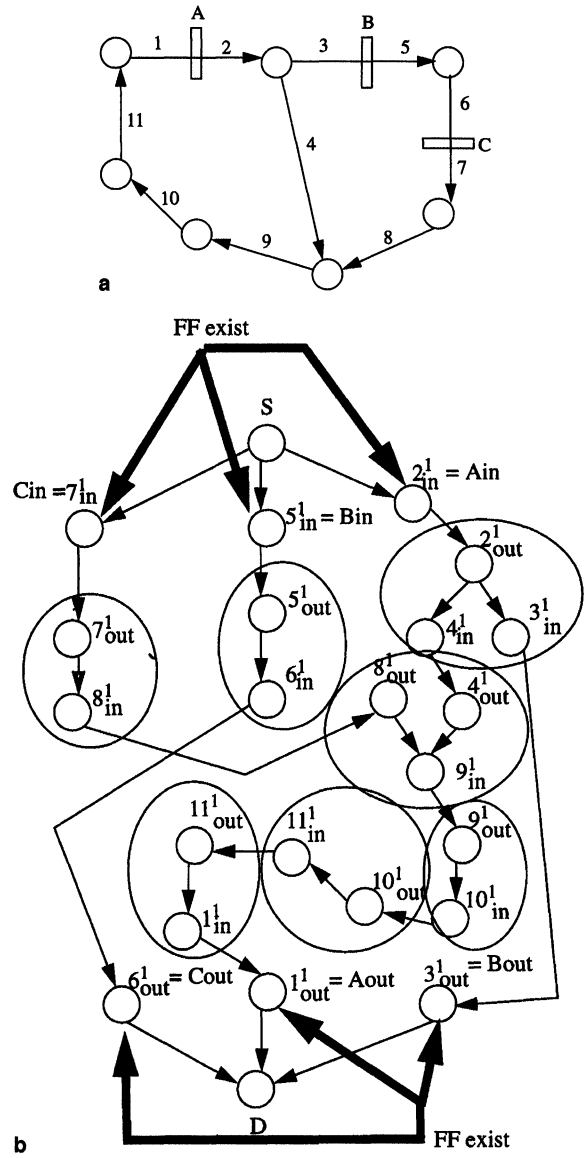

FIGURE 3 The straightforward approach for sequential circuits. (a) A sequential circuit. (b) The directed acyclic graph that corresponds to the circuit of (a). Every register R is replaced by two modules, R_{in} and R_{out}. All edges from S to any R_{in} and from any R_{out} to D have 0 weight. Similarly, all edges from any R_{in} to any module, and from any module to any R_{out} have 0 weight. The rest of the construction is identical to the one for combinational circuits.

Leiserson and Saxe [9] proposed algorithms for clock period minimization, for both circuits with uniform and nonuniform delays on the modules. We found the proposed algorithm for circuits with

nonuniform delays on the modules [9] to be complicated to implement. In this paper, we present an approach for the general model, which although asymptotically has the same time complexity with the respective algorithm in [9], is faster in practice and much easier to implement.

The formulation of our problem requires a retiming technique with some constraints on top of the ones in [9]. More precisely we perform retiming on a graph constructed as follows: Every module's input and output is represented by a node, and every internal edge is substituted by a node and two edges labeled as prohibitive. An edge is called *prohibitive* if it is not allowed to host any flip-flop. After this transformation, all nodes have uniform delays. (see also Fig. 4). Our problem is to do retiming on a cyclic graph $G=(V, E)$, constructed from the sequential circuit so that we minimize the clock period subject to a set of prohibitive edges E_p. It can be shown that the problem is equivalent to assigning weights $r(u)$ on the nodes $u \in V$, that satisfy the following conditions:

C1: $r(u) - r(v) \leq w(e)$, $\forall e=(u, v) \in E$.

C2: $r(u) - r(v) \leq W(u,v) - 1$, $\forall u,\ v \in V$ such that $D(u, v) > c$.

C3: $r(u) - r(v) = w(e)$, $\forall e=(u, v) \in E_p$, where E_p is the set of prohibitive edges.

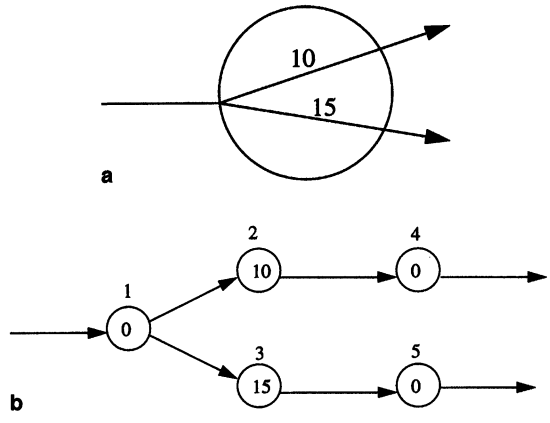

FIGURE 4 The transformation using the prohibitive edge model. (a) A functional element with nonuniform delays. (b) The element of (a) after the transformation. The labels inside the circles indicate uniform delays on the respective nodes.

C1, C2 guarantee clock period minimization in [9]. C3 guarantees that no flip-flop is placed on a prohibitive edge.

Next, we present a more detailed description of the heuristic. First we substitute every (p, q)-module by $(pq+p+q)$ modules whose delay is uniform. All p_i, q_j modules have delay 0. If max $(p, q) > s$, s is a small constant (we set in our experiments $s=3$), we consider the (p, q)-module as a module with uniform delay equal to the maximum delay among all I/O paths. We follow this type of approach to speed up operations. We call this model the *prohibitive-edge model.* The rest of the graph construction is the same as the one described in the straightforward approach earlier. Thus, the created graph is a DAG, and we can select the module with the biggest gain by employing our proposed approach for combinational circuits. In a second step, the heuristic changes the created DAG to a graph with cycles, by considering registers. Then it performs retiming on the new graph, by applying the algorithm for clock period minimization, algorithm OPT2 in [9], that uses the prohibitive-edge model, described above. Thus the second step runs in $O(|V||E|\log|V|)$ time. The heuristic repeats the two steps until no feasible retiming exists. The time complexity is therefore $O(|V|(|V||E|+|V||E|\log|V|)) = O(|V|^2|E|\log|V|)$.

For comparison reasons, we also implemented a variation of the previously described approach. In this version, although we create a graph using the prohibitive-edge scheme described above, we don't break any cycles. Furthermore, we perform retiming once to obtain the minimum clock period before any module changes have taken place. Then we calculate the gain of each module by performing retiming, using algorithm [9]. The gain of a module is equal to the difference of the minimum clock period evaluated before any module changes minus the minimum clock period of the circuit considering the alternative implementation of the particular module. The module with the biggest gain is selected and locked. Thus, for every module change, the heuristic performs retiming once for

each unchanged module. The time complexity is $O(|V|^3|E|\log|V|)$.

4. EXPERIMENTAL RESULTS

We implemented both our approach and the straightforward iterative improvement for the combinational circuits in C and run on a Sun Sparc System 4/330. We experimented on several ISCAS'85 benchmarks. Since the ISCAS'85 circuits do not include a list of possible implementations for each module, we generated these randomly by using function rand() from the standard library. For simplicity reasons, we considered uniform pin capacitances and the simplified model. We applied mod 10 to all created numbers, so that the delays range from 0 to 9. We treat every cell from the library as a module. Table I, gives the experimental results for our heuristic, Comb1, and the straightforward iterative improvement approach Comb2. In Table I, "initial delay" denotes the initial delay of the circuit's longest path, "delay" the minimum delay obtained for the longest path, and "time" the time required for the particular heuristic to terminate. The time here is expressed in seconds.

When we constructed Comb1, we tried to stay as close to the gain computation as possible, expecting to get a little worse results than those of Comb2 (since the gains were computed approximately) but faster. We observed that in 80% of the gain selection Comb1 did indeed select the best actual gain. From Table I, observe that Comb1 is not only much faster than Comb2 as expected, but is also produces smaller delays. A possible explanation of this behavior is that the suboptimal gains helped escaping local minima.

We implemented our heuristics for the sequential circuits in C and run on a Sun Sparc System 2. We experimented on several ISCAS'89 benchmarks. For simplicity reasons, we run our three heuristics based on the assumption that all modules have uniform delays. We used the delays given by the ISCAS'89 circuit data as the delays of the first implementation. We generated the delays of the modules for the second implementation randomly using the function rand() from the standard library. In addition, we applied mod 10 to all generated numbers so the delays range from 0 to 9. As before, we treat each cell from the library as a module. Table II, presents the experimental results for the straightforward approach, Seq 1, our heuristic, Seq 2, and Seq 3. Under "initial delay", we give the initial delay of the circuit's longest path. Under "delay", we list the minimum delay obtained for the longest path. Under "time", we give the time required for the particular heuristic to terminate. The time is expressed in seconds. Moreover, under "initial min.clock" we give the initial minimum clock period obtained by using retiming in Seq 2, Seq 3, before any module swaps have taken place. Although retiming appears to be relatively slow

TABLE I Results for Comb 1 and Comb 2

Circuit	# of nodes	initial	Comb 2		Comb 1	
ISCAS		delay	delay	time	delay	time
C17	9	26	22	0.02	22	0.007
C432	196	110	99	190	93	4.2
C499	243	72	68	698	59	11.2
C880	443	140	119	3222.3	102	28.7
C1355	587	160	151	4941	126	66.2
C1908	913	206	181	14527.4	170	126.9
C2670	1350	184	155	39670	141	866.6
C3540	1719	236	198	24921	184	423.3
C5315	2485	240	191	57645	179	2478.7
C6288	2448	642	532	49781	507	1162.7
C7552	3719	230	188	71542	174	4535.4

TABLE II Results for Seq 1, Seq 2 and Seq 3

Circuit ISCAS	# of nodes	initial delay	Seq 1		Seq 2		Seq 3		initial min.clock
			delay	time	delay	time	delay	time	
s208.1	122	114	52	0.62	22	78.2	20	7287.5	44
s298	137	102	40	0.9	33	96.1	30	12297.5	48
s349	185	220	87	1.4	67	156.3	64	25037	156
s382	182	114	57	2.3	39	144	35	29743.1	64
s386	172	110	66	1.6	48	112.1	46	20581.4	110
s444	204	134	64	5.6	47	467	46	40453.3	68
s510	236	136	52	9.4	51	612	51	74316.8	126
s838.1	512	174	142	45	134	3867	132	341895	152

process, we were able to obtain good results by using the prohibitive edge scheme described in Section 4. The results were in practice faster than the approach in [9] for the general model (with the simplifications described in the prohibitive edge model), by a factor of approximately 10%. Seq 3 was inapplicable even for small circuits. More importantly, the latter modification did not lead to any improvements on the quality of the obtained implementations.

For simplicity reasons we implemented all of our heuristics without considering any area constraints. Note though that all of our heuristics can be trivially modified in order to handle a given upper bound on the total area of the circuit. The idea is to select the module that has the highest gain but whose selection does not violate the area upper bound. When the module with the biggest gain has been obtained, the particular heuristic checks whether by considering this module the overall area of the circuit exceeds the given area bound. If the area bound is exceeded, then we discard the module and we proceed to the module that has the highest gain among the remaining ones. Otherwise, the module is selected.

5. CONCLUSION

We have shown that the general circuit implementation problem is NP-hard in the strong sense even when each module has only two implementations and there is no constraint on the total area of the circuit. We call this problem the TDGCI problem. The BCI problem, where all paths in a module have the same delays but different implementations have different areas is also NP-hard in the strong sense even for two implementations per module.

We proposed the first heuristic for combinational circuits under the general circuit implementation model. The heuristic uses iterative improvement methodology and outperforms an alternative iterative improvement scenario. We also proposed a retiming based heuristic for sequential circuits which uses the one for combinational circuits as a subroutine. The approach is compared to two other schemes we devised.

Acknowledgement

Research supported by NSF grant MIP-9409905.

References

[1] Brayton, R. K., Hachtel, G. D. and Sangiovanni-Vincentelli, A. L. "Multilevel logic synthesis", *Proc. of the IEEE*, **78**(2), pp. 264–300, February 1990.

[2] Chan, P. K. (1990). "Algorithms for library-specific sizing of combinational logic", *27th ACM/IEEE Design Automation Conference* (DAC '90), pp. 353–356.

[3] Chaudhary, K. and Pedram, M. (1992). "A Near Optimal Algorithm for Technology Mapping Minimizing Area under Delay Constraints", *Proc. 29th ACM/IEEE Design Automation Conference* (DAC '92), pp. 492–498.

[4] Cormen, T. H., Leiserson, C. E. and Rivest, R. L. (1990). Introduction to Algorithms, MIT Press, Cambridge MA.

[5] Garey, M. R. and Johnson, D. S. (1979). Computers and Intractability, W. H. Freeman and Co., New York.

[6] Karayiannis, D. G. and Tragoudas, S. (1995). "TIMING-DRIVEN CIRCUIT IMPLEMENTATION", Technical Report 94-04, Computer Science Department, Southern Illinois University, Oct. 1994. The paper appears in part at the Proceedings of the 5th Great Lakes Symposium on VLSI, pp. 2–7.

[7] Keutzer, K. (1987). "DAGON: technology binding and local optimization by DAG matching", *Proc. 24th ACM/IEEE Design Automation Conference* (DAC'87), pp. 341–347.
[8] Lawler, E. L., Levitt, K. N. and Turner, J. (1969). "Module Clustering to Minimize Delay in Digital Networks", *IEEE Trans. on Computers*, pp. 47–57, c18, no 1.
[9] Leiserson, C. E. and Saxe, J. B. (1991). "Retiming Synchronous Circuitry", *Algorithmica*, **6**, pp. 5–35.
[10] Li, W. N. (1993). "Strongly NP-hard Discrete Gate Sizing Problems", *Proc. IEEE Int. Conf. Computer Design* (ICCD '93), pp. 468–471.
[11] Li, W., Lim, A., Agrawal, P. and Sahni, S. (1992). "On The Circuit Implementation Problem", *IEEE Trans. on CAD*, **12**(8), pp. 1147–1156, 1993. A preliminary version appears in the Proceedings of the 27th ACM/IEEE Design Automation Conference (DAC '92), pp. 478–483.
[12] Murgai, R., Brayton, R. K. and Sangiovanni-Vincentelli, A. (1991). "On Clustering for Minimum Delay/Area", *Proceedings of the IEEE International Conference on Computer-Aided Design* (IC-CAD '91), pp. 6–9.
[13] Pedram, M. and Bhat, N. (1991). "Layout driven logic restructuring/decomposition", *Proc. IEEE Int. Conf. Computer-Aided Design* (ICCAD '91), pp. 134–137.
[14] Rajaraman, R. and Wong, D. F. (1993). "Optimal Clustering for Delay Minimization", *Proceedings of the 30th ACM/IEEE Design Automation Conference* (DAC '93), pp. 309–314.
[15] Sherwani, N. (1993). Algorithms for VLSI Physical Design Automation, Kluwer Academic Publishers MA.
[16] Touati, H. J., Moon, C. W., Brayton, R. K. and Wang A. (1990). "Performance-oriented technology mapping", *Proc. 6th MIT Conf., Advanced Research in VLSI*, W. J. Dally ed., pp. 79–97.
[17] Touati, H. J., Savoj, H. and Brayton, R. K. (1991). "Delay optimization of combinational logic circuits by clustering and partial collapsing", *Proc. IEEE Int. Conf. Computer Design* (ICCD '91), pp. 188–191.
[18] Weste, N. H. E. and Eshraghian, K. PRINCIPLES OF CMOS VLSI DESIGN, Addison-Wesley Publishing Company, 2nd Edition.

APPENDIX

We show that the BCI problem is NP-complete in the strong sense. A somewhat stronger but more complex reduction was also given in [10] for the same problem. We reduce from a restricted version of the ONE-IN-THREE 3SAT problem [5] which we call the RESTRICTED ONE-IN-THREE 3SAT, and is also NP-complete [5]:

RESTRICTED ONE-IN-THREE 3SAT
Input: Set U of variables, collection C of clauses over U such that each clause $c_k \in C$, $1 \leq k \leq |C|$ has $|c_k| = 3$, and does not contain any negated literal.
Question: Is there a truth assignment for U such that each clause in C has exactly one true literal?

THEOREM 1 *The BCI decision problem is NP-complete even if the maximum number of possible implementation for each module is two.*

Proof Clearly, the BCI decision problem is in NP. Next, we transform RESTRICTED ONE-IN-THREE 3SAT to the BCI decision problem. Let $U = \{u_1, u_2, \ldots, u_n\}$ be a set of variables and $C = \{c_1, c_2, \ldots, c_m\}$ be a set of clauses making up an instance of RESTRICTED ONE-IN-THREE 3SAT. We shall construct an instance of BCI such that the RESTRICTED ONE-IN-THREE 3SAT instance is satisfiable if and only if the constructed BCI instance has an implementation with area at most A, and delay at most D. We obtain the BCI instance as follows.

Let l_i^k be the i^{th} literal of clause c_k. In the RESTRICTED ONE-IN-THREE 3SAT problem no $c_k \in C$ contains a negated literal. Thus, each literal l_i^k is equal to some u_j, where $u_j \in U$. For every variable, u_i of the RESTRICTED ONE-IN-THREE 3SAT instance, we construct a module, called variable module, and labeled U_i. Each variable module is an (m, m)-module, where m is the number of clauses in RESTRICTED ONE-IN-THREE 3SAT instance. Let the i^{th} input and i^{th} output of module U_j be labeled U_j^i and $U_j'^i$, respectively. The structure of a variable module, as well as the labeling of its inputs and outputs is given in Figure 5a.

The variable modules are connected as follows: Let l_1^k, l_2^k, l_3^k be the first, the second, and the third literals of clause c_k, respectively. Assume that $l_1^k = u_i$, $l_2^k = u_j$, and $l_3^k = u_r$, where $u_i, u_j, u_r \in U$. We connect $U_i'^k$ with U_j^k, and $U_j'^k$ with U_r^k.

We now describe the two possible implementations for each variable module, as well as an upper bound A on the area and an upper bound D on the delay. The latter bounds, together with our construction, guarantee that each clause $c_k \in C$ has exactly one true literal if and only if the area is at most A, and the delay is at most D. More precisely, let m_i be the number of clauses that

contain variable u_i. Clearly $\Sigma_i m_i = 3m$. Every variable module U_i has two implementations. In the first implementation, the area is 0 and the delay is 1. In the second implementation, the area is m_i and the delay is 0. (See also Fig. 5b). If u_i is assigned the value true, then we assign module U_i its first implementation. If u_i is assigned the value false, then we assign its second implementation. Note that the way we construct the variable modules guarantees a consistent true/false assignment on the variables. In addition, we set the area A to be $2m$, and the delay D to be 1. An example of the BCI instance corresponding to a RESTRICTED ONE-IN-THREE 3SAT instance is shown in Figure 5b.

We now show that the RESTRICTED ONE-IN-THREE 3SAT instance is satisfiable if and only if the constructed BCI instance has an implementation with area at most A, and delay at most D. We call clause path C_k, the path along the variable modules for the variables corresponding to the clause c_k (see Fig. 5b). Observe that the clause paths are the longest paths among any input-output paths. In fact, all the remaining input-output paths consist of one edge, and can have delay at most 1. If the RESTRICTED ONE-IN-THREE 3SAT instance is satisfied, then each clause $c_k \in C$ has one true and two false literals. Therefore, the delay on every clause path is the same and equal to 1, and the area of the whole BCI instance is $2m$. Thus, the BCI instance is satisfied.

On the other hand, suppose that the constructed BCI instance is satisfied. Then the delay along any input-output path is at most 1, and the area of the BCI instance is at most $2m$. Next, we show that the delay along any clause path is exactly 1, and the area of the BCI instance is exactly $2m$.

Assume, by contradiction, that one of the clause paths has delay 0. This means that all three variables of the corresponding clause are assigned the value false, and therefore, the corresponding variable modules are assigned their second implementation. Let these three variables be u_i, u_j, and u_r. From the construction, it follows that modules U_i, U_j, and U_r contribute to the overall area of the BCI instance m_i, m_j, and m_r units of area, respectively. Moreover in order to satisfy the upper bound on the delay constraint, every other clause must have at least two variables assigned the value false. Thus, besides the clause whose three variables are evaluated to be false, all the other clauses must have at most one variable evaluated to be true. Thus, in the RESTRICTED ONE-IN-THREE 3SAT instance there are less than m variables which are evaluated to be true. The latter implies that the overall area exceeds the bound of $2m$, a contradiction.

Therefore no clause path can have delay 0, and all clause paths have delay exactly 1. The latter implies that exactly one variable per clause is true. Since our construction guarantees consistent true/false assignment to the variables, we conclude that

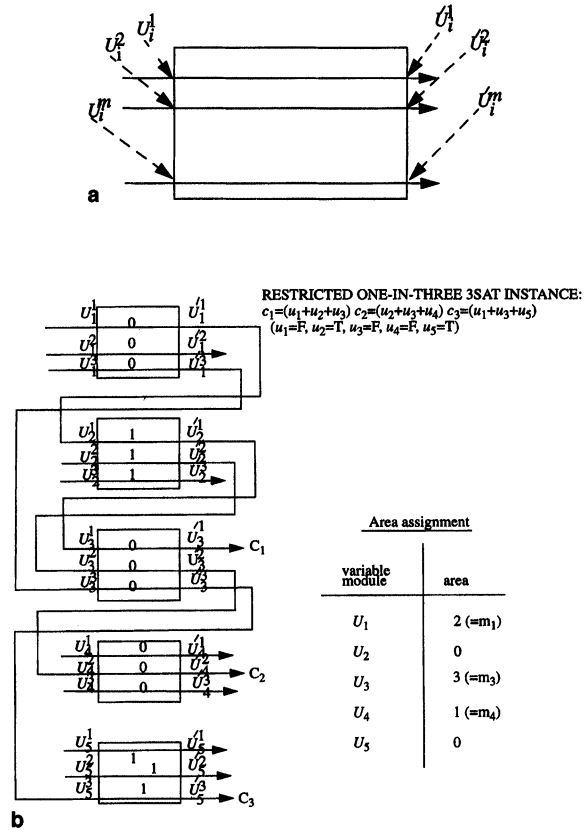

FIGURE 5 The reduction for the BCI problem. (a) Variable module U_i. (b) The BCI instance.

the RESTRICTED ONE-IN-THREE 3SAT instance is satisfied. □

Authors' Biographies

Dimitrios G. Karayiannis was born in Greece, on March 12, 1969. He received the B.S. degree in Computer Science from Southern Illinois University in 1991 and the M.S. degree in Computer Science from Southern Illinois University in 1993. Currently. He received his Ph.D. degree in Computer Science, Southern Illinois University in 1996.

From 1991 to 1993, he was a teaching assistant. From 1993 to 1996 was a research assistant in the department of Computer Science, Southern Illinois University. Since 1997 has been with Viewlogic Systems Inc. His research interests include Computer Aided Design (algorithms and applications), Design for Testability, Test Pattern Generation, and Built-In Self-Test.

Spyros Tragoudas received his Diploma degree in Computer Engineering from the University of Patras, Greece (July 1986) and his M.S. and Ph.D. degrees in Computer Science from the University of Texas at Dallas (August 1988 and August 1991, respectively). He joined the faculty of Southern Illinois University at Carbondale in August 1991, where he is currently an Associate Professor of Computer Science.

His research interests include VLSI Testing, Computer Aided Design for Physical Design Automation, and algorithms for combinatorial optimization problems. In 1993, he received the Research Initiation Award from the National Science Foundation, MIPS Division, for research on VLSI Testing. He also received (together with D. Kagaris) the ICCD'94 Outstanding Paper Award, Design and Test Truck, for a paper on LFSR-based BIST Test pattern Generation. Dr. Tragoudas is a member of IEEE Computer and CAS societies.