Minimum Area Retiming with Equivalent Initial States [1]

Naresh Maheshwari
Department of Electrical & Computer Engineering
Iowa State University, Ames IA 50011, USA
naresh@iastate.edu

Sachin S. Sapatnekar
Department of Electrical & Computer Engineering
University of Minnesota, Minneapolis, MN 55455, USA
sachin@ece.umn.edu

Abstract

Traditional minimum area retiming algorithms attempt to achieve their prescribed objective with no regard to maintaining the initial state of the system. This issue is important for circuits such as controllers, and our work addresses this problem. The procedure described generates bounds on the retiming variables that guarantee an equivalent initial state after retiming. A number of possible sets of bounds can be derived, and each set is used to solve a minimum area retiming problem that is set up as a 0/1 mixed integer linear program, using a new technique that models the maximal sharing of flip-flops at latch outputs. The best solution is found through enumeration of these sets, terminated on achievement of a calculated lower bound. Experimental results show that after a small number of enumerations, optimal or near-optimal results are achievable.

1 Introduction

Retiming [1] is a technique for optimizing sequential circuits by relocating memory elements. Two common variations of this problem are: *minperiod* retiming in which the clock period is minimized without regard to the number of flip-flops (FF's) in the final circuit, and (constrained) *minarea* retiming in which the number of FF's is minimized subject to a target clock period. Retiming has also been applied to level-clocked circuits e.g. in [2, 3].

One problem associated with the application of retiming is preserving the initial state of the circuit, which is determined by the initial values of the registers in the circuits. Whenever the initial state of the circuit is an integral part of its behavior (for example, in controllers), it is necessary to find an equivalent initial state for the retimed circuit. An initial state in the retimed circuit is equivalent to the initial state in the original circuit if for any input sequence applied to both the circuits (original circuit started in the initial state and the retimed circuit started in the equivalent initial state) the same sequence of outputs is produced [4].

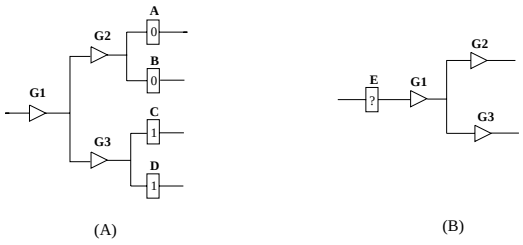

Figure 1: (A) Original circuit. (B) Retimed circuit

It is not always possible to find an equivalent initial state for the retimed circuit. For example in Figure 1(a), if the initial value of FF's {A,B,C,D} are {0,0,1,1} respectively, then the retimed circuit in Figure 1(b) cannot be initialized to have the same behavior as the original circuit. This is because an equivalent initial value of FF E in the retimed circuit cannot be found. On the other hand, an equivalent initial state can always be found for forward motion of FF's (i.e., in the direction of signal flow). This concept was used in [5] to compute initial states of retimed circuits by using only forward moves. FF's could be removed from all primary outputs and inserted at all primary inputs, and the problem was reduced to determining the initial values for the inserted FF's; their values could be obtained from the state machine description of the circuit. However, this approach may require modifications in the combinational logic which may increase the clock period.

Since a retiming for a given clock period is not unique, another possible retiming may exist, for which an equivalent initial state can be found without any circuit modifications. Reverse retiming [4] finds this retiming by disallowing FF moves across the primary outputs and by minimizing backward (against the signal flow) motion.

For digital circuit design, the useful objective function is that of constrained minimum area retiming. However, none of the above methods considers the area penalty during retiming since they perform minperiod retiming rather than minarea retiming. In this work we solve the problem of minarea retiming with equivalent initial states and call it *minarea initial state retiming*. We use bounds on the retiming variables to allow backward motion of FF's only if an equivalent initial value exits. Therefore any retiming thus obtained is guaranteed to have an equivalent initial state. Retiming across the host vertex is not allowed since it may require modifications to the original circuit. We also provide a new formulation that takes into account the initial value of the FF's while modeling the maximal sharing of FF's at the outputs of multiple fanout gates.

2 Background

As in [1] a sequential circuit can be represented by a directed graph $G(V, E)$, where each vertex v corresponds to a gate, and a directed edge e_{uv} represents a connection from the output of gate u to the input of gate v, through zero or more FF's. Each edge has a weight $w(e_{uv})$, which is the number of FF's between the output of gate u and the input of gate v. Each vertex has a constant delay $d(v)$. A special vertex, the host vertex, is introduced in the graph, with edges from to all primary inputs of the circuit, and edges from all primary outputs. A retiming is a labeling of the vertices $r : V \rightarrow Z$, where Z is the set of integers. The retiming label $r(v)$ for a vertex v represents the number of FF's moved from its output to its inputs.

The minarea retiming problem (without regard to initial states) was formulated as a linear program (LP) in [1]. Let $FI(v)$ [$FO(v)$] be the fanin [fanout] sets of gate v respectively. The objective function in the LP represents the number of FF's added by retiming, and the constraints ensure a valid retimed circuit that satisfies the target clock period. This LP is a dual of a network mincost flow problem and hence can be solved efficiently. Minaret [6] reduces the size of this LP by adding lower and upper bounds on the variables; the details are omitted here. The reduced LP has the form:

$$\begin{array}{lll} \text{minimize} & \sum_{v \in V} [(|FI(v)| - |FO(v)|) \cdot r(v)] & (1) \\ \text{subject to} & r(u) - r(v) \leq c_{uv} & \forall \text{ constraints } \in C \\ & L_u \leq r(u) \leq U_u & \forall u \in V \end{array}$$

where C is the reduced constraint set [6].

[1] This work was supported in part by the National Science Foundation under award MIP-9502556 and a Lucent Technologies DAC Graduate Scholarship.

3 Ensuring Equivalent Initial States

The requirement of initial state equivalence imposes restrictions in addition to those in traditional minarea retiming. Thus the number of FF's obtained in minarea retiming is, by definition, a lower bound on the number of FF's obtainable by a minarea equivalent state retiming. We call this lower bound Γ.

However it is not always possible to achieve this lower bound. As an example, consider the circuit with unit delay gates shown in Figure 1. The minarea retiming for a clock period of 2 units, and without regard to initial state requires only 1 FF. However, if the initial values of FF's {A,B,C,D} are {0,0,1,1}, any minarea initial state retiming will require 2 FF's. Furthermore, the optimal number of FF's depends on the initial state of the original circuit. If the initial values for FF's {A,B,C,D} are {0,1,1,1}, then any minarea initial state retiming will have 3 FF's.

Even in cases where the lower bound Γ is achievable with equivalent initial states, there would, in general, be multiple retimings with optimal number of FF's. Some of these retimings may not have equivalent initial states, and hence we must restrict our solution space to exclude such solutions. One way to do this is to disallow backward motion of FF's across a gate if the FF's at its output do not have compatible values. The presence of FF's with incompatible logic values at the output of a gate is called a *conflict*. A conflict at the output of a gate prevents it from being retimed in the backward direction. Therefore to ensure that any retiming obtained has an equivalent initial state we update the upper bound U_v on gate v in the LP of Equation (1), so that no backward retiming is allowed across a gate with a conflict at its outputs. This new upper bound $J_v \leq U_v$ ensures a valid equivalent state. Notice that we do not update the lower bounds since the forward motion of FF's always results in an equivalent initial state.

In the remainder of this section, we will describe techniques to obtain these new upper bounds. Minaret [6] obtains the upper bounds by moving FF's backwards until they are about to violate a period constraint; the number of the FF's moved across any gate gives its upper bound. Unlike Minaret which does not associate any logic values with the FF's, we associate a three valued (1,0,X) logic value with every FF, where X is a don't care which can be assigned to either 0 or 1. A logic value of 0 [1] is compatible with both 0 and X [1 and X], but logic values 0 and 1 are not compatible with each other.

A gate can only be retimed backwards if it has FF's at all of its fanouts and all of these FF's have compatible logic values. The procedure maintains a list of gates that can be retimed. In each step, a gate is plucked from the list and retimed, and the list is updated. Whenever FF's are moved from the outputs of a gate to its inputs, we must assign logic values to the new FF's added at the inputs. These logic values must be equivalent to the original value at the gate output in order to maintain state equivalence. This assignment may be unique or non-unique and is similar to justification in ATPG [7].

Unique justification at a gate occurs if the gate has a single input, or the logic value at the output is X (all inputs are assigned to logic X). A logic value of 1 at the output of AND/NOR gates or logic 0 at the output of OR/NAND gates also results in unique justifications. If there are multiple possible mappings for the logic value at the output to the logic values at the inputs, then we have to make a choice (or decision) and we have *non-unique justification* at the gate. A logic value of 1 at the output of an OR (NAND) gate is an example of non-unique justification and we can assign any input to logic value 1(0) and the rest to X.

Let us define a justification set as a backward propagation of logic values until the primary inputs, or until a conflict is reached. Note that non-unique justifications at gates permit a number of such justification sets; we denote one such possible justification set as Δ^i. Then each such Δ^i will give us Λ^i, a set (one for each gate) of justification upper bounds. For each Δ^i we solve the minarea LP with the upper bounds Λ^i. If the number of FF's so obtained is not equal to the minarea lower bound Γ, we backtrack and obtain another justification set Δ^j to give us a different Λ^j. This process can be repeated until the minarea lower bound Γ is achieved or until all justification sets have been enumerated.

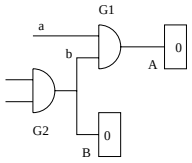

Figure 2: An example of pruning technique

The number of justification sets to be explored is exponential in the worst case; however, this number can be pruned by removing suboptimal Λ^i's from consideration. As an example, consider the circuit in Figure 2 with the logic values of FF A and FF B equal to 0. Since the output of the AND gate G1 is at logic 0, there are two possible mappings for the equivalent values at a and b. However the choice of setting input a to X and input b to 0 is better than the choice of $a = 0$ and $b =$ X, since the presence of FF B with logic 0 will force the X on line b to 0. Thus we make the choice of $a = 0$ and $b = 0$, which is suboptimal to the choice of $a =$ X and $b = 0$, since X on input a could potentially move further in the backward direction than a 0. For another pruning strategy, consider two sets of justification upper bounds Λ^i and Λ^j. If we have $J_v^i \leq J_v^j \ \forall\ v \in V$, then the feasible region of Λ^j includes the feasible region of Λ^i, and hence there is no need to solve the LP corresponding to Λ^i.

4 FF Sharing

The LP in Equation (1) assumes that the FF's at the fanouts of a gate are not shared amongst the different fanouts. However, to accurately model the minimum number of FF's in a circuit, we must take maximal FF sharing into account. In [1] a mirror vertex is added for every gate with more than one fanout to model this maximal sharing. This model preserves the LP's duality to a mincost flow problem, and can be solved efficiently. Unfortunately, this model assumes that an FF can be combined with any other FF, and hence is not applicable to minarea initial state retiming where FF's have logic values associated with them, and an FF with logic 1 can not be shared with one with logic value 0. The situation is complicated by the fact that two FF's can be shared only if the FF's at their fanins (if any) are also shared. We present a new 0/1-MILP formulation to model the sharing when FF's have logic values 1 or 0. This modeling is used for all gates with a conflict at their output. For all other gates the simpler model of [1] is used. We will first present the model and then illustrate it through an example.

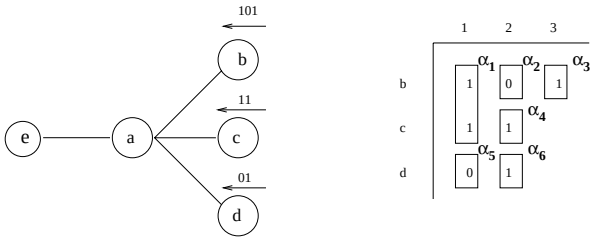

Figure 3: An example for FF sharing

The justification process of Section 3 determines the logic values of all FF's that can possibly arrive at a gate's fanouts. Notice that there is a sequence of these "possible" FF's at every output of every gate, and the final retiming may contain only a prefix of this sequence. The logic values of these possible FF's at the fanouts of a gate u are represented by a table T_u with $|FO(u)|$ rows as shown

in Figure 3, where $FO(u)$ is the fanout set of gate u. Each row, $v \in FO(u)$ has $J_v + w(e_{uv})$ entries, each of which is either a 0 or a 1. Since a maximum of J_v FF's can be moved across gate v to its input, and $w(e_{uv})$ FF's already exist between gate u and gate v, the maximum number of FF's between gate u and gate v that may require conditional sharing is $J_v + w(e_{uv})$. FF's moved forward across gate u to its output can be shared unconditionally and will be handled later. The value in the v^{th} row and k^{th} column of the table is denoted by $T_u(v,k)$. We define a sharing class S_i to contain a set of values that can be shared, and represent the set of sharing classes for the fanouts of gate u by N_u. Two values (p,q) and (r,s) in T_u can be shared (i.e., belong to the same sharing class) only if $q = s$ and $T_u(p,i) = T_u(r,i)$ for $i = 0, \cdots, s-1$. A function $class(T_u(v,k))$ gives the index of the sharing class for entry (v,k) in table T_u, e.g., $S_{class(T_u(v,k))}$ is the sharing class containing the k^{th} FF between gate u and its fanout v (counting from u). All the FF's in a sharing class can be shared with each other, and hence require only one physical FF. Each sharing class S_i is represented in the MILP by a variable $\alpha_i \in \{0,1\}$. If $\alpha_i = 1$ in the optimal solution of the MILP, then the FF's of sharing class S_i share a physical FF and the sharing class S_i is said to be active.

The minarea LP in Equation (1) is modified to model the conditional sharing represented by the sharing classes. For every gate u with a conflict at its outputs, the corresponding objective function term is given by Equation (2); for all other gates it is as in [1].

$$(|FI'(u)| - 1) \cdot r(u) - \beta_u + \sum_{\forall\ i \in N_u} \alpha_i \qquad (2)$$

Here $FI'(u)$ is the set of fanins that have only a single output, i.e., $FI'(u) = \{v | v \in FI(u)\ AND\ |FO(v)| = 1\}$. The first term $(|FI'(u)|-1) \cdot r(u)$ in Equation (2) models the increase in the number of FF's when gate u is retimed by one unit, and is similar to the model in [1]. It assumes a shared cost of one at the fanouts of gate u for any set of FF's retimed across gate u, in either direction. Since a gate can be retimed backwards only if all FF's at its output have same logic values, the shared cost at the outputs before retiming is indeed one, as modeled by this term. Notice that since $r(u) \leq J_u$, no set of FF's with shared cost greater than one, can ever be retimed backwards across gate u. In forward retiming, all FF's inserted at the outputs of a gate have the same logic values, and so the shared cost at fanouts of gate u in forward retiming is also one. The second term $\beta_u \geq 0$ is a correction factor applied to correctly model the situation in which a set of FF's moves forward across gate u and all its fanouts. It is active only during forward retiming steps, and models the number of FF's removed from the fanout junction of gate u by forward retiming. $\beta_u \geq 0$ ensures that unconditionally shared FF's are not added to the fanouts of gate u by backward retiming, since conditional sharing under backward retiming is to be modeled by the α_i's.

As mentioned earlier $\alpha_i = 1$ implies that the sharing class S_i is active, therefore $\sum_{\forall\ i \in N_u} \alpha_i$ denotes the number of active sharing classes at the fanouts of gate u. Since each active sharing class requires one FF, the number of active sharing classes is also the number of physical FF's required at the fanouts of gate u. The minimization of the objective function will force the maximal sharing at the outputs of gate u.

In order to correctly model the cost, all the FF's at the fanout of a gate u must have their costs accounted for in the objective function. To achieve this we add the following constraint for $\forall\ v \in FO(u)$.

$$w(e_{uv}) + r(v) \leq -\beta_u + \sum_{k=1}^{J_v + w(e_{uv})} \alpha_{class(T_u(v,k))} \qquad (3)$$

The left hand side of Equation (3), minus $r(u)$, is the number of FF's between gate u and v after retiming. $\sum_{k=1}^{J_v+w(e_{uv})} \alpha_{class(T_u(v,k))}$ represents the total number of active sharing classes between gate u and v, β_u is the number of FF's removed from the fanout of gate u by forward retiming (0 in case of backward retiming), while $r(u)$ is the number of FF's removed from the fanout of gate u. Hence the right hand side of Equation (3), minus $r(u)$, represents the number of available shared FF's between gate u and v. Thus Equation (3) ensures that the number of FF's between any two gates is less than or equal to the number of shared FF's between them. Notice that the number of shared FF's is the number of available FF's , all of which do not have to be utilized by a particular fanout.

To ensure that the k^{th} FF retimed across gate v activates its own sharing class variable $\alpha_{class(T_u(v,k))}$, we ensure that $\forall v \in FO(u)$

$$\left\{\alpha_{class(T_u(v,k))} \geq \alpha_{class(T_u(v,k+1))}\right\}\ k = 1 \ldots J_v + w(e_{uv}) - 1$$

By requiring that the variable $\alpha_{class(T_u(v,k))}$ be active before the variable $\alpha_{class(T_u(v,k+1))}$, this constraint ensures that the k^{th} FF retimed across gate v does not activate $\alpha_{class(T_u(v,k+1))}$. The k^{th} FF retimed across gate v can not activate $\alpha_{class(T_u(v,k-1))}$, since before the k^{th} FF can be retimed across gate v, the $k-1^{th}$ FF must have already been retimed and its class variable $\alpha_{class(T_u(v,k-1))}$ would already be active. The constraint in Equation (3) will ensure that the number of active sharing classes on a fanout is greater or equal to the number of FF's on that fanout. Thus the k^{th} FF retimed across gate v will activate its own sharing class. The first FF in a sharing class S_i that arrives at the fanout junction activates the sharing class variable α_i, incurring a cost of one in the objective function. The remaining FF's in that sharing class can then arrive without incurring any extra cost in the objective function.

Example: Consider the circuit in Figure 3 shown with the sharing classes in its table of logic values. The LP for this circuit is

$$Minimize: -r(b)-r(c)-r(d)+\alpha_1+\alpha_2+\alpha_3+\alpha_4+\alpha_5+\alpha_6-\beta_a$$

$$\begin{aligned}
\text{subject to} \quad & r(b) + \beta_a \leq \alpha_1 + \alpha_2 + \alpha_3 \\
& r(c) + \beta_a \leq \alpha_1 + \alpha_4 \\
& r(d) + \beta_a \leq \alpha_5 + \alpha_6 \\
& \alpha_1 \geq \alpha_2 \geq \alpha_3 \\
& \alpha_1 \geq \alpha_4\ ;\ \alpha_5 \geq \alpha_6 \\
& \beta_u \geq 0\ ;\ \alpha_i \in \{0,1\}\ \ \forall i
\end{aligned}$$

Positive Retiming: Suppose we want to model the sharing for $r(a) = 0$, $r(b) = 3$, $r(c) = 1$ and $r(d) = 2$. Then the optimal objective function value of the above LP is -1, which gives the correct increase in the number of FF's from the original circuit in Figure 4(a) to the retimed circuit in Figure 4(b).

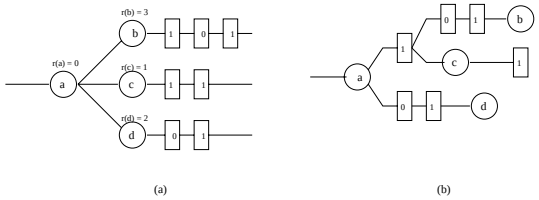

Figure 4: Example of positive retiming

Negative retiming: Now suppose we want to model the sharing for $r(a) = -2$, $r(b) = -2$, $r(c) = -1$ and $r(d) = -1$. Then the optimal objective function value is 3, which is the increase in the number of FF's from the original circuit in Figure 5(a) to the retimed circuit in Figure 5(b). As can be seen one FF is shared for the edges e_{ac} and e_{ad} even though they where not in the same sharing class. This is possible because the FF's moved forward to the outputs of gate a hence they all have same logic value without regard to the sharing

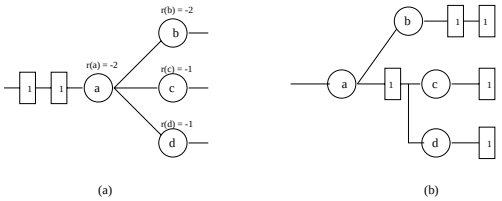

Figure 5: Example of negative retiming

class which are defined for backward movements. Thus these FF's can be shared and our formulation correctly models the cost.

The actual problem of FF sharing is to find the optimal sharing between logic values 0, 1 and X. Logic X can be shared with either 0 or 1 but not both, and is hard to model. To avoid this problem we have converted all X's to either 0 or 1 before formulating the MILP.

5 Experimental Results

We have implemented an initial state minarea retiming based on the presentation in this work. Our justification algorithm makes random choices in case of a non-unique justification and generates an LP for each of the Λ_i's. If the lower bound Γ is not achieved, then we perform a justification based on another random decision. This is continued until the lower bound is reached or a user specified number of iterations have been completed, and the best solution is reported. Although it may seem arbitrary to use random decisions, our experimental results show that the algorithm gives us good engineering solutions that are close to the (possibly unachievable) lower bound. As in [4] we assume the initial state of all FF's to be zero.

Table 1: Minarea Initial State Retiming

Circuit	$\|G\|$	P	Γ	# FF's	T_{exec}
s27	11	6.0	3	3	0.01s
s208.1	105	10.0	8	8	0.02s
s298	120	6.0	22	22	0.40s
s382	159	7.0	23	23	2.59s
s386	169	11.0	6	6	0.04s
s344	161	14.0	19	19	1.77s
s349	162	14.0	19	19	1.62s
s526n	195	6.0	30	30	0.95s
s510	212	11.0	7	7	0.12s
s420.1	219	12.0	17	17	0.07s
s641	380	74.0	19	19	0.11s
s713	394	74	19	19	0.18s
s967	395	12.0	35	35	28.52s
s938	447	16.0	33	33	1.45s
s1196	530	24.0	18	18	0.08s
s1238	5.09	22.0	18	18	0.08s
s1269	570	19.0	84	84	0.26s
s1423	658	53.0	76	76	8.77s
s1488	654	16.0	7	7	0.11s
s1494	648	16.0	7	7	0.13s
s3330	1790	14.0	110	110	0.58s
s5378	2780	21.0	173	173	3m 18s
s9234.1	3271	38.0	134	134	21m 18s
s635	287	66.0	35	42	22.6s
s953	396	13.0	27	32	32m 02s
s1512	781	23.0	70	71	1h 51m 19s
s3271	1573	15.0	168	169	16m 46s
prolog	1602	13.0	122	124	16m 40s
s3384	1686	27.0	167	168	55m 42s
s15850.1	9618	63.0	525	544	3h 9m 56s

If for a given justification set Δ_i there are no gates with conflicts, the mincost flow problem is solved using a network simplex algorithm; else we use *lp_solve* [9] to solve the MILP.

Table 1 shows the number of gates $|G|$, the target clock period P, and the lower bound on the number of FF's Γ for ISCAS89 circuits. We also show the minimum number of FF's obtained with equivalent initial state retiming and the execution time T_{exec} on a HP 9000/777 C110 workstation with 128 megabytes of RAM. Notice that the run times here can be much higher then those in [6] since here we may solve multiple MILP's, rather than a single mincost flow problem. As can be seen from the results, for many circuits the lower bound is achieved very fast; in almost all of these cases, the lower bound Γ is achieved in the first iteration. For some circuits, the lower bound was not reached, and in this case we report the best solution obtained in 50 iterations (5 iterations for s15850.1)[2]. In these circuits, the solution reported by our algorithm is very close to Γ and corresponds to a good engineering solution. We found that the percentage of gates with conflicts is very small ($< 1\%$ for most circuits), and this enables the MILP to be solved in reasonable time.

6 Conclusion

We have presented a method for obtaining minarea retiming subject to maintaining a given clock period and an equivalent initial state. A new scheme based on justification is used to derive bounds on the retiming variables. A new model for maximal FF sharing has been presented as the idea of mirror vertices used by Leiserson and Saxe in [1] cannot be applied to the initial state retiming problem.

We are currently working on modeling the FF sharing for all logic values including X's and on techniques to prune the number of justification sets required to get the optimal solution. In [10] a technique is presented which allows backward retiming of gates with conflicts by adding extra logic to the circuit. We are investigating means to incorporate this approach in our model. This will enable us to retime gates with conflicts at their outputs, potentially achieving better area optimization.

REFERENCES

[1] C. E. Leiserson and J. B. Saxe, "Retiming synchronous circuitry," *Algorithmica*, vol. 6, pp. 5–35, 1991.

[2] M. C. Papaefthymiou and K. H. Randall, "TIM: A timing package for two-phase, level-clocked circuitry," *Proc. DAC* , pp. 497–502, 1993.

[3] N. Maheshwari and S. S. Sapatnekar, "A practical algorithm for retiming level-clocked circuits," in *Proc. ICCD*, pp. 440–445, 1996.

[4] G. Even, I. Y. Spillinger, and L. Stok, "Retiming revisited and reversed," *IEEE Transactions on Computer-Aided Design*, vol. 15, pp. 348–357, Mar. 1996.

[5] H. J. Touati and R. K. Brayton, "Computing the initial states of retimed circuits," *IEEE Transactions on Computer-Aided Design*, vol. 12, pp. 157–162, Jan. 1993.

[6] N. Maheshwari and S. S. Sapatnekar, "An improved algorithm for minimum-area retiming," in *Proc. DAC*, pp. 2–7, 1997.

[7] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital Systems Testing and Testable Design*. New York, NY: W. H. Freeman and Company, 1990.

[8] S. Kundu *et al.*, "A small test generator for large designs," in *Proc. ITC*, pp. 30–40, 1992.

[9] M. Berkelaar, *LP_SOLVE USER'S MANUAL*, 1992.

[10] V. Singhal, S. Malik, and R. K. Brayton, "The case for retiming with explicit reset circuitry," in *Proc. ICCAD*, pp. 618–625, 1996.

[2] The reader is reminded from our previous discussion that the lower bound is not guaranteed to be achievable.