Retiming Control Logic

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

Retiming is a powerful technique for delay and area optimization that operates by relocating the flip-flops in a circuit. This movement of flip-flops in control logic changes the state encoding of finite state machines, requiring the preservation of initial (reset) states. Unfortunately traditional retiming algorithms pay no regard to maintaining the initial state. While some work has been carried out on finding a retiming of a circuit with equivalent initial states, it has concentrated on achieving a specified clock period without regard to the number of flip-flops. However, if the number of flip-flops is not explicitly minimized the retimed circuit may have a very large number of flip-flops. This work targets the problem of minimizing the number of flip-flops in control logic subject to a specified clock period and with a guarantee of an equivalent initial state. The problem is formulated as a mixed integer linear program and bounds on the retiming variables are used to guarantee an equivalent initial state. These bounds also lead to a simple method for calculating an equivalent initial state for the retimed circuit. The mixed integer linear program formulation is capable of modeling the maximum sharing of different types of flip-flops at the fanout of a gate. Experimental results on circuits of up to 9000 gates and are shown to be close to a (perhaps unachievable) lower bound.

1 Introduction

Retiming is a procedure that involves the relocation of flip-flops (FF's) across logic gates to allow the circuit to be operated under a faster clock. The technique was first proposed by Leiserson and Saxe [1, 2]. The retiming problem as posed by Leiserson and Saxe was framed in the following ways:

(1) The **minimum period (minperiod)** retiming problem where FF's are relocated to obtain a circuit with the minimum clock period, without any consideration to the area penalty due to an increase in the number of FF's. Retiming for a specified clock period is a special case of this problem.

(2) The **constrained minimum area (minarea)** retiming problem, where FF's are relocated to achieve a given target clock period with the minimum number of FF's. Retiming to minimize the number of FF's without any regard to the period is a special case of this problem.

Several research efforts have extended the Leiserson-Saxe method to handle variations of the original problem, for example, retiming level-clocked circuits [3, 4], improving the delay model [5, 6], retiming for low

power [7,8], and combining retiming with synthesis [9,10]. The work in [11] presented results on the validity of retiming sequential circuits.

One problem associated with the application of retiming is related to the preservation of the initial (reset) state of a circuit, which is determined by the initial values of the registers in the circuits. In the synthesis of control logic, the initial state of the circuit is an integral part of its behavior. Therefore, it is necessary to find an equivalent initial state for the retimed circuit. An initial state in the retimed circuit is equivalent to that in the original circuit if for any input sequence applied to both the circuits, with the original circuit started in the initial state and the retimed circuit started in the equivalent initial state, the same sequence of outputs is produced [12].

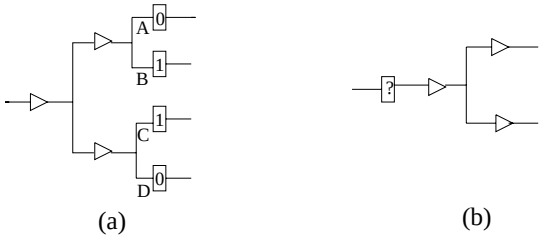

Figure 1: (a) Original circuit. (b) Retimed circuit

It is not always possible to find an equivalent initial state for the retimed circuit without modifying it. For example, consider the circuit in Figure 1 taken from [13]. If the initial value of FF's {A,B,C,D} are {0,0,1,1}, respectively, then the retimed circuit cannot be initialized to have the same behavior as the original circuit since an equivalent initial value of FF C in the retimed circuit cannot be found. Techniques for finding a retiming with an equivalent initial state were proposed in [12,13].

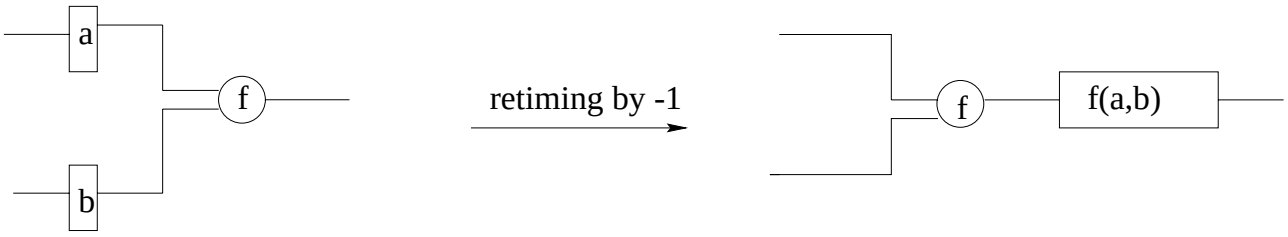

Figure 2: Forward retiming of a combinational logic node

As shown in Figure 2, an equivalent initial state can always be found for forward motion of FF's (referred to as "negative retimings" using the notation of Leiserson and Saxe). Thus, one way to ensure that an equivalent initial state can always be found is to permit only forward retiming moves. This concept was used by Touati and Brayton in [13] to compute initial states of retimed circuits. In their approach, FF's may be removed from all the primary outputs and inserted at all the primary inputs, corresponding to a motion across the host node (defined in Section 2.1). The problem is then reduced to determining the initial values for the FF's inserted at the primary inputs.

Permitting only forward moves is too restrictive because some backward moves have equivalent initial states. For example if, in the circuit in Figure 1 both FF A and B have the same initial value, then the backward move across gate G1 is possible while maintaining equivalent initial state. Hence, another retiming

requiring some backward moves may exist, that enables one to find an equivalent initial state without any modifications to the circuit, even for circuits where the method of [13] required circuit modifications. Reverse retiming [12, 14] finds this retiming by disallowing FF moves across the primary outputs and by minimizing their backward motion.

For digital circuit design, the most useful objective function is that of constrained minarea retiming. However none of the above methods considers the area penalty during retiming to achieve the target clock period since they perform minperiod retiming rather than minarea retiming. The standard minarea algorithms, e.g., [2, 15, 16], pay no regard to the initial states. While they have applications in datapaths where the initial state is unimportant, they cannot be used to optimize control logic since an equivalent initial state is not guaranteed to exist in the retimed circuit.

We believe that this work is the first to target the problem of minarea retiming for control logic guaranteeing equivalent initial states. As in [13], we use the phrase *retiming an initial state* to mean finding a retiming (with an initial state) such that the original circuit and the retimed circuit have the same behavior when started in their respective initial states. We use the term *minarea initial state retiming* to refer to retiming an initial state with minimum number of FF's. As in most of the references on retiming listed above, this work assumes the circuit to be composed of gates with fixed delays, and considers circuits with edge-triggered FF's.

In this research, which we believe to be the first published work on the problem of optimal 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 will have an equivalent initial state. There may be multiple sets of these bounds, and all of them must be explored to obtain an optimal minarea initial state retiming. However the number of FF's obtained by standard minarea retiming can be used as a lower bound to prune this exploration.

This work also provides a new minarea initial state formulation that takes into account the initial value of the FF's while modeling the sharing of FF's at the output of a multi-fanout gate. We will show through an example in Section 4.2 that the pre-justification approach, where we use justification choices before retiming to build the FF sharing model, is essential to obtain a correct FF sharing model; in the absence of this method, the FF cost computed by the traditional FF sharing model is incorrect in the presence of initial conditions.

The method presented here is applicable for retiming of any circuit which has more than one type of memory elements such that memory elements of different types can not be merged together. An example is a situation of FF's with load enables that can only be combined if the enabling signal to two FF's is the same.

The rest of the paper is organized as follows: Section 2 presents the required background and the standard minarea retiming algorithm. Next in Section 3 we present an method to ensure the existence of an equivalent initial state, followed by the FF sharing model in Section 4. We present experimental results in Section 5 and conclude the paper in Section 6.

2 Background

2.1 Notation

The notation in this section is reproduced from [2] and is presented for completeness. 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, one 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 $d(v)$. The fanin and the fanout set of a vertex v are denoted by $FI(v)$ and $FO(v)$ respectively. A special vertex called the host vertex H, is introduced in the graph, with edges from the host vertex to all primary inputs of the circuit, and edges from all primary outputs to the host vertex.

A retiming is a labeling of the vertices $r : V \to Z$, where Z is the set of integers. The weight of an edge e_{uv} after retiming, denoted by $w_r(e_{uv})$ is given by

$$w_r(e_{uv}) = r(v) + w(e_{uv}) - r(u) \tag{1}$$

The retiming label $r(v)$ for a vertex v represents the number of FF stages moved from its output towards its inputs. One may define the weight $w(p)$ of any path $p : u \rightsquigarrow v$ originating at vertex u and terminating at vertex v, as the sum of the weights on the edges on p, and its delay $d(p)$ as the sum of the weights of the vertices on p. A path with $w(p) = 0$ corresponds to a purely combinational path with no FF's on it; therefore, the clock period can be calculated as

$$c = \max_{p|w(p)=0} \{d(p)\} \tag{2}$$

An important concept used in the Leiserson-Saxe approach is that of the W and D matrices that are defined for all pairs of vertices (u, v) such that there exists a path $p : u \rightsquigarrow v$ that does not include the host vertex. $W(u, v)$ denotes the minimum latency, in clock cycles, for the data flowing from u to v, and $D(u, v)$ gives the maximum delay from u to v for the minimum latency, i.e.,

$$W(u, v) = \min_{p:u;v} \{w(p)\} \tag{3}$$

$$D(u, v) = \max_{p:u;v \text{ and } w(p)=W(u,v)} \{d(p)\} \tag{4}$$

2.2 The minarea retiming problem

The conventional minarea retiming problem (without regard to initial states) was formulated in [2] as the following linear program (LP):

$$\begin{aligned} & \text{minimize} && \textstyle\sum_{v \in V} (|FI(v)| - |FO(v)|) \cdot r(v) && && (5) \\ \text{subject to} \quad & && r(u) - r(v) \le w(e_{uv}) && \forall e_{uv} \in E \\ & && r(u) - r(v) \le W(u, v) - 1 && \forall D(u, v) > c \end{aligned}$$

where $|FI(v)|$ and $|FO(v)|$ represent the number of fanins and fanouts of a gate v respectively, and c, as in the earlier discussion, is the target clock period.

Recently the Minaret algorithm [16] combined the ASTRA approach of [17] with the minarea retiming method of [15] to achieve efficient minarea retiming. In this method the LP (5) was reduced by adding bounds on the r variables.

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 , .

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 , 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.

Thus one way to ensure that any obtained retiming has an equivalent initial state is to update the upper bounds U_v in the Minaret formulation [16], so that conflicting FF's at the fanouts of a gate are never retimed to its inputs. This will ensure a valid equivalent state in the retimed circuit. This new upper bound on gate v that ensures a valid equivalent state is called B_v. Since we want a retiming that has an equivalent state *and* satisfies the target clock period we need to enforce $r(v) \leq B_v$ and $r(v) \leq U_v$. If we define *justification upper bound* as $J_v = \min(B_v, U_v)$, then we only need to ensure $r(v) \leq J_v$. A set of such justification upper bounds denoted by $\Lambda^i = \{J_u^i | \ \forall u \in V\}$. Since forward retiming moves always have equivalent initial values, the lower bounds from Minaret for conventional minarea retiming are still valid for minarea initial state retiming. Thus we obtain the following modified LP

$$\begin{array}{lll} \text{minimize } \sum_{v \in V} [(|FI(v)| - |FO(v)|) \cdot r(v)] & & (6) \\ \text{subject to} \quad r(u) - r(v) \leq c_{uv} & \forall c(u,v) \in C & \\ L_u \leq r(u) \leq J_u & \forall u \ \in V & \end{array}$$

Any solution to this LP will have an initial state that is equivalent to the initial state in the original circuit and will satisfy the target clock period. The techniques of [16] can be applied to further reduce the size of the LP (6).

3.1 Obtaining the Justification Bounds

We will now describe a method for obtaining these new justification upper bounds J_v for a gate v. The procedure consists of two steps: a justification step, where an equivalent initial state is found, and a bound computation step, where the bounds J_v on each gate under that equivalent initial state are calculated.

With every FF we associate a three valued (1,0,X) logic value. We define *compatibility* as follows: a logic value of 0 is compatible with both 0 and X, but logic values 0 and 1 are not compatible with each other[1]. A gate can only be retimed if it has FF's with compatible logic values at all of its fanouts. A gate is retimed in the backward direction by removing an FF from each of its fanouts, and adding one to each of its inputs. A gate is called output-ready if it has an FF on each of its fanouts and the logic value on each such FF is compatible with the values on the others. The procedure maintains a list of gates that can be retimed. A gate is taken from the list and retimed, and the list is updated. As the gates are retimed, a procedure similar to [16] is used to compute the bounds. The upper bounds, J_v are obtained by moving FF's as far backwards as possible without violating the period constraints. The count of the FF's moved across any gate gives its upper bound on the r variable of the gate.

Each time 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 output of the gate in order to obtain an initial state retiming. This assignment, in general, may not be unique and is similar to the phase of justification in the process of automatic test pattern generation [18]. *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), e.g., a logic value of 1 at the output of AND gates requires a logic value of 1 at all the inputs. If there are multiple possible mappings for the logic value at the output to the logic values at the inputs, then we must 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 gate is an example of non-unique justification as we can assign any input to logic value 1 and the rest to X.

3.2 Searching for the Optimal Solution

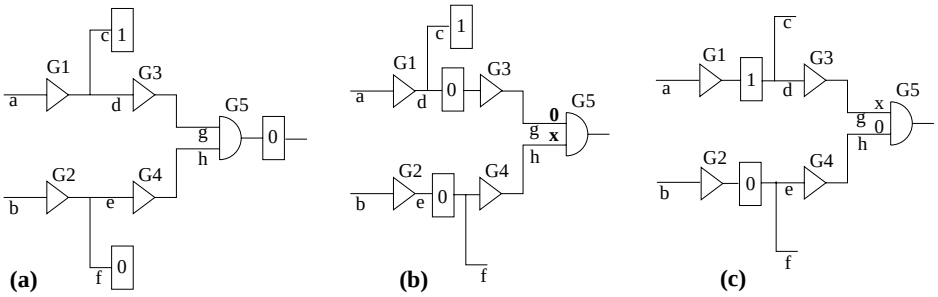

Figure 3: Effect of justification on the number of FF's

Different justification decisions may lead to a different number of FF's obtained after minarea retiming. As an example, consider the circuit shown in Figure 3(a), with an FF with value 0 at the output of an AND

[1]For circuits with multiple types of memory elements that cannot be combined, compatibility can be defined similarly.

gate, leading to two possible choices shown in Figures 3(b) and 3(c). The corresponding decisions lead to retimed circuits with three and two FF's, respectively.

Under nonunique justifications, a number of different allowable justifications are possible. Let us define a set of one such possible justification as Δ^i. Each such Δ^i will give us a set (one for each gate) of justification upper bounds $\Lambda^i = \{J_v^i | \ \forall v \in \ V\}$ that is used to solve the minarea LP. If the number of FF's so obtained is not equal to the minarea lower bound , , we must backtrack and obtain another set of justifications Δ^j that leads to a different Λ^j. This process is repeated until we either achieve the minarea lower bound , , or no more justifications exist. Since a complete exploration will be computationally expensive, one may halt the exploration of the search space at any time and take the best solution obtained so far.

Thus the process of minarea initial state retiming can be given by the following pseudocode. The procedure returns the minarea retiming with an equivalent initial state.

```
1     Obtain minarea lower bound ,
2     j = 0;
3     Best = ∞;
4     while (true)
5     {
6       while (U ≠ ∅ OR D ≠ ∅)
7       {
8         if (U ≠ ∅) do_unique_justification
9         if (D ≠ ∅) = do_decision_justification
10      }
11        /* This gives us a justification set Δ^j.  */
12        /* which corresponds to a set of justification upper bounds Λ^j.  */
13      Obj = lp_solve(Λ^j); /* solve LP in (6) */
14      If(Obj == , ) return(Obj); /* lower bound obtained */
15      If(Best > Obj) Best = Obj; /* store best result */
16      B = backtrack(Δ^j);
17      If(B == Infeasible) return(Best); /* all justifications explored */
18    }
```

The function `backtrack` changes the last decision that has a yet unexplored choice, and is similar to one used in automatic test pattern generation (see for example [19]). The period constraints need be generated only once during the entire procedure since they do not depend on the justification process. This is helpful since the period constraint generation is a very computationally intensive process.

The theoretical upper bound on the number of possible justification sets Δ^i's in the worst case is $|FF| \cdot \prod_{\forall v \in V} |FI(v)|$, where $|FI(v)|$ is the number of fanins of the gate v, and $|FF|$ is the number of FF's in the original circuit. This upper bound is due to the fact that in the worst case, each FF in the circuit may move across every gate and every such move may require a decision. However in practical circuits the number of feasible justifications will be much less than this theoretical upper bound due to the following reasons

- As shown in the results of [16] the mobility of FF's is very limited in practice, and hence, all FF's cannot move across all gates as assumed by the theoretical bound above.
- Due to conflicts at the gate fanouts the FF's may not be able to move towards the inputs of that gate, and this further restricts the mobility of the FF's.
- Some FF's moving across gates have unique justifications.
- Every time a decision is made in case of a nonunique justification, all fanins but one are assigned logic value X for AND/OR/NAND/NOR type of gates). This logical X moves backward through unique justification until it is forced to a 0 or 1.
- As soon as the lower bound of , is achieved we do not need any more justification sets. In our experimental results we found that in many circuits this lower bound is achieved in the first few iterations.
- Only backward moves need justification, while forward moves have a unique mapping of logic values, and hence, do not add to the number of Δ^i's.

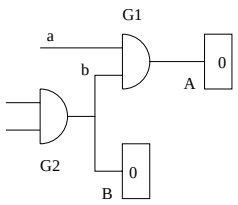

Figure 4: An example of pruning technique

The number of justification sets can be further reduced by pruning suboptimal Λ^i's. Consider the circuit in Figure 4 with the logic values of FF A and FF B equal to 0. Since the output of the AND gate G1 is at logic value 0 we have two possible mappings for the equivalent values at the inputs 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 $b = X$. This is because in the presence of FF B with logic value 0, the X on input b will be forced to a 0, effectively setting the choice to $a = 0$ and $b = 0$. This is suboptimal to the choice of $a = X$ and $b = 0$, since X on input a can move further back than a 0.

4 FF Sharing

The cost function in the LP (6) assumes that each FF has exactly one fanout. However, in practice an FF can have multiple fanouts, allowing the FF's on different fanout edges of a gate to be shared. This sharing must be taken into account for an accurate area model. For example, consider gate A in Figure 5 with three fanouts B, C, and D having three, two and two FF's respectively. The LP (6) will model the total number of FF's as seven as shown in Figure 5(a). However the FF's can be merged or shared as shown in Figure 5(b) resulting in a total cost of only three FF's.

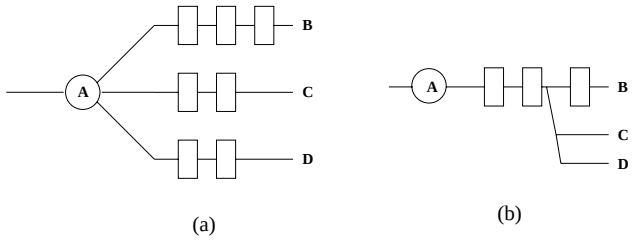

Figure 5: Unconditional register sharing at multiple fanouts

We first briefly describe the sharing model from [2], the details of which can be found in [20]. This model does not consider initial values on the FF's, and hence is not applicable for minarea initial state retiming. We then present a new model for conditional sharing of FF's, which takes into account the initial values on the FF's.

4.1 Unconditional Sharing of FF's

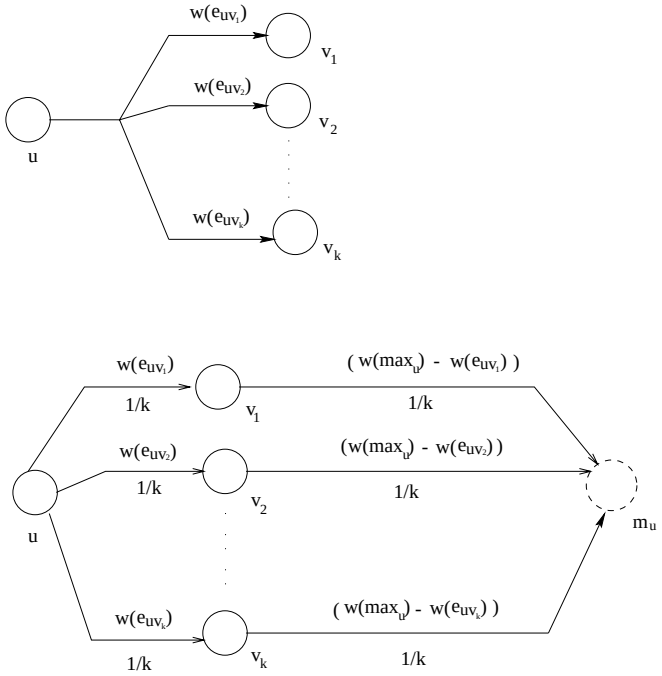

Figure 6: Model for maximum register sharing at multiple fanouts

The unconditional sharing model of [2] introduces a mirror vertex m_i for each gate i that has more than one fanout, as shown in Figure 6. Every edge e_{ij}, in addition to having a weight $w(e_{ij})$, now also has a width $\beta(e_{ij})$. In Figure 6, the edge weights are shown above the edges while the edge widths are shown below the edges. Consider a gate u with k fanouts to gates v_j, $j = 1 \cdots k$. To model the maximum sharing of FF's, an extra edge is added from each fanout gate v_j to the mirror vertex, m_u, with weight $w(e_{v_j m_u}) = w(max_u) - w(e_{uv_j})$, where $w(max_u) = \max_{\forall i \in FO(u)}(w(e_{ui}))$ is the maximum weight on any

fanout edge of gate u. Each of the edges from the gate i to its fanouts j, and from the fanouts to the mirror vertex has a width of $1/k$, i.e.,

$$\beta(e_{uv_j}) = 1/k \text{ and } \beta(e_{v_j m_u}) = 1/k \quad \text{for } j = 1 \cdots k$$

. The objective function of the LP (6) is modified to include the effect of register sharing as follows:

$$\min \sum_{v \in (V \cup M)} \left[\left(\sum_{\forall j \in FI(v)} \beta(e_{jv}) - \sum_{\forall j \in FO(v)} \beta(e_{vj}) \right) \cdot r(v) \right]$$

$$\begin{aligned} \text{subject to} \qquad & r(u) - r(v) \leq c_{uv} \;\; \forall (u,v) \in C \\ & r(j) - r(m_i) \leq c_{jm_i} \;\; \forall (j, m_i) \in C_m \end{aligned} \tag{7}$$

where $M = \{m_v | v \in V \text{ and } |FO(v)| > 1\}$ is the set of all the mirror vertices, and $C_m = \{(j, m_i) | m_i \in M \text{ and } j \in FO(i)\}$ is the set of constraints due to the mirror vertices and is called the mirror constraint set. The weight c_{jm_i} on a constraint $(j, m_i) \in C_m$ of the form $r(j) - r(m_i) \leq c_{jm_i}$ is given by $c_{jm_i} = w(max_i) - w(e_{ij})$.

The objective function of the LP (7) now denotes the increase in the number of FF's assuming maximal sharing of FF's at the output of all gates. The weights on all paths from gate u to its mirror vertex m_u are the same before retiming, i.e., $w(e_{uv_i}) + w(e_{v_i m_u}) = w(max_i) \;\; 1 \leq i \leq k$, and therefore, the weights on all paths from gate u to its mirror vertex m_u must be equal after retiming. Since the mirror vertex m_u is a sink in the graph, the register count on one of the edge from the fanout nodes to m_u will be zero, i.e., $\exists i \;\; |w(e_{v_i, m_u}) = 0$. Thus the weight on all paths from gate u to mirror vertex m_u after retiming will be $w_r(max_u) = \max_{\forall j \in FO(u)}(w_r(e_{uj}))$. Since there are k paths each with width $1/k$ the total cost of all paths will be $w_r(max_u)$ as desired.

We now present an alternate view of this model. The change in cost function due to adding or removing FF's from the fanout junction of gate u is modeled by two retiming variables: one for the gate, $r(u)$ and other for the mirror vertex, $r(m_u)$. Any change in the cost function due to FF's moving across the multi-fanout gate itself are modeled by $r(u)$, while any change due to FF motion across its fanout gates $v_i, \;\; 1 \leq i \leq k$ is modeled by the mirror variable $r(m_u)$.

The change in the number of FF's in the circuit, under maximal sharing obtained by retiming a gate u by one unit can be calculated as follows. The decrease in the cost function obtained by removing an FF from each of the fanouts of a gate is one unit, even for multiple fanout gates since the FF's on all the fanouts were shared. The increase in the cost function from adding an FF to all the inputs of a gate u is equal to the number of fanins of u that have only one fanout, since any FF added to a fanin j of gate u that has more than one fanout ($|FO(j)| > 1$) is already modeled by the mirror variable of that fanin gate m_j. Thus the cost contribution of any single fanout gate u is given by $(|FI'(u)| - 1) \cdot r(u)$, while that of a multi-fanout gate is given by $(|FI'(u)| - 1) \cdot r(u) + r(m_u)$, where $FI'(u)$ is the set of fanins that have only a single output, i.e., $FI'(u) = \{v | v \in FI(u) \; AND \; |FO(v)| = 1\}$.

4.2 Motivation for Conditional FF Sharing

Consider the possibility of performing minarea initial state retiming by extending the approach in [12] using post-justification after successive retiming steps, instead of pre-justification before performing the retiming, as we do with the FF sharing model that will shortly be proposed.

This extension would first perform conventional minarea retiming, and if a conflict occurs at a gate while moving the FF's to obtain this retiming in a post-retiming justification stage, then an appropriate bound is placed on the retiming variable of this gate. The minarea retiming problem may be solved again, and the process repeated until no conflicts are obtained. Thus the final circuit has an equivalent initial state although it may require more FF's than the conventional minarea, since the bounds placed on the retiming variables to ensure equivalent initial states can increase the optimal solution. This method can be seen as a "dual" of our approach, since it starts from the lower bound and tries to achieve feasibility (equivalent initial state), while in our approach we start with a feasible solution and try to achieve optimality. However, using this approach without a conditional FF sharing model can lead to suboptimal results, as will be shown in the following example.

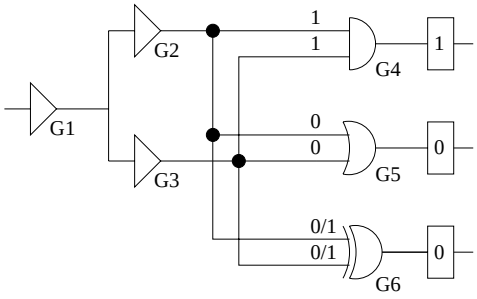

Figure 7: Conditional register sharing at multiple fanouts

Consider the circuit in Figure 7. Unconstrained minimum area retiming with no initial state considerations will find an optimal solution by moving the FF's to the output of gate G1, resulting in one FF. However, this is an infeasible location with the given set of initial states as the values cannot be justified backwards due to conflicts at gates G2 and G3. Let us now say that we apply an extension of the approach in [12] by detecting these conflicts and updating their r bounds to prohibit FF movements to the left across these two gates, and using the Leiserson/Saxe FF sharing model. The solution now found will be a two-FF solution with one FF each at the outputs of G2 and G3, placed before the fanout point. However, due to initial state conflicts (the justified values are shown in the figure, with both possible justifications shown for the XOR gate), the number of FF's must actually be 4, corresponding to two FF's at the outputs of each of G2 and G3, with initial states of 0 and 1. This solution is clearly suboptimal, and the optimal solution is the original configuration, with 3 FF's.

Therefore, the use of a Leiserson-Saxe like FF sharing model with post-justification is likely to result in suboptimal solutions. This motivates the value of using the FF sharing model with pre-justification that we propose in the next section. It is very important to note that this example shows that the use of post-justification will not lead to the optimal solution and hence pre-justification is essential for correctly counting the number of FF's.

4.3 Conditional FF sharing

The model in Section 4.1 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 value 1 can not be shared with one that has logic value 0. For example, consider the circuit in Figure 5(a) with initial state values as shown in Figure 8(a). With these initial values, the sharing given by the mirror vertex model of Section 4.1 is shown in Figure 5(b); this is not valid for the given initial values. Instead, the maximal sharing is as shown in Figure 8(b) and requires a total six FF's. The reason is that only two FF's, shown in the dashed box in Figure 8(a), can be shared. The situation is further complicated by the fact that two FF's can be shared only if the FF's at their fanins (if any) are also shared. For example, consider the circuit in Figure 8(b), the FF's on output C and D cannot be shared, although both have an initial state value of 1, because their fanins are not shared.

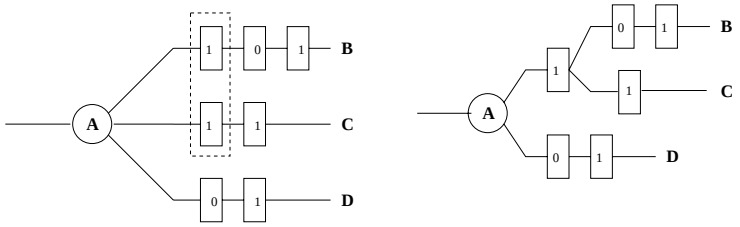

Figure 8: Conditional register sharing at multiple fanouts

Thus we need a way to model the conditional sharing when FF's have initial values associated with them. This conditional sharing is also required for circuits that have more than one type of FF's that can not be shared with each other. We will now present the modifications required to model the conditional sharing by a 0/1 mixed integer linear program (0/1 MILP) formulation. This modification is used for all gates with conflicts at their fanouts, and for all other gates the simpler model of [2] is used. This combination keeps the number of integer variables within a small fraction of the total number of variables. We will first present the model and then illustrate it through an example.

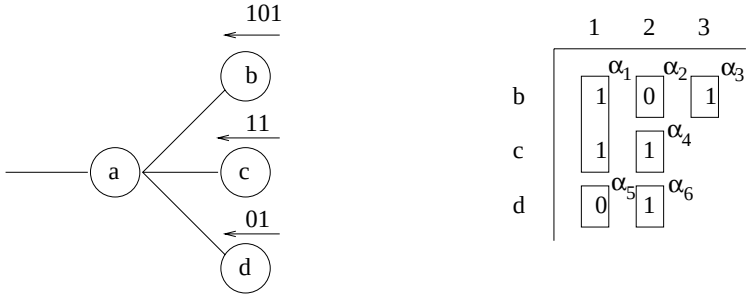

Figure 9: An example for FF sharing

The justification process of Section 3 determines the logic values of all FF's that can possibly be retimed backwards to arrive at the fanout of a given gate. There is a sequence of these "possible" FF's that may

arrive at every fanout of every gate, and possibly be retimed across the gate, or remain at the gate output; the final retiming may contain only a subsequence of this possible 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 9. Since a maximum of J_v FF's can be moved backwards across gate v to its fanins, and $w(e_{uv})$ FF's already exist between gate u and gate v, the maximum number of FF's possible between gate u and gate v is $J_v + w(e_{uv})$. Therefore, each row, $v \in FO(u)$, has $J_v + w(e_{uv})$ entries, each of which is either a 0 or a 1.

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[2] 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 $T_u(p,q)$ and $T_u(r,s)$ 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. FF's moved forward across gate u to its fanouts can be shared unconditionally and will be handled later.

To ensure that the k^{th} FF retimed across gate v activates its own sharing class variable $\alpha_{class(T_u(v,k))}$, we require that the variable $\alpha_{class(T_u(v,k))}$ be active before the variable $\alpha_{class(T_u(v,k+1))}$. This is achieved by adding the following constraint

$$\alpha_{class(T_u(v,k))} \geq \alpha_{class(T_u(v,k+1))} \;\; \forall v \in FO(u) \text{ and } 1 \leq k \leq J_v + w(e_{uv}) - 1$$

For every multi-fanout gate u we also define an integer variable $\rho_u \leq 0$, which models the forward retiming. This is required because unlike backward retiming, FF's introduced at the fanouts by forward retiming across gate u can be unconditionally shared since all of them have the same logic value. Thus the α variables model the backward retiming and ρ models the forward retimings. Notice that this is different from the unconditional sharing model of Section 4.1 where the mirror variable $r(m_u)$ modeled FF's moved by both forward and backward retimings since all FF's could be unconditionally shared. Requiring the α variables to be nonnegative, $\alpha_i \geq 0$ ensures that they model only forward (positive) retiming moves, while the condition $\rho_u \leq 0$, ensures that ρ_u models only backward (negative) retiming moves.

4.3.1 Modifications in the Objective Function

To model the conditional sharing represented by the sharing classes, the objective function term for a gate u that has a conflict at its fanouts is modified to

$$(|FI'(u)| - 1) \cdot r(u) + \rho_u + \textstyle\sum_{i \in N_u} \alpha_i \tag{8}$$

This expression counts the number of FF's that settle at the output of gate u after retiming and the significance of each term is as follows:

[2]Sharing classes for circuits with different types of FF's can be defined similarly.

- The first term $(|FI'(u)| - 1) \cdot r(u)$ in (8) models the increase in the number of FF's when gate u is retimed by one unit, and is similar to the model in [2]. As earlier $FI'(u)$ is the set of fanins that have only a single output, i.e., $FI'(u) = \{v | v \in FI(u)\ AND\ |FO(v)| = 1\}$. It assumes a shared cost of one at the fanouts of gate u for any set of FF's retimed in either direction across gate u. In forward retiming, all FF's inserted at the outputs of a gate have the same logic values, and therefore, the shared cost at fanouts of gate u in forward retiming is one. Since a gate can be retimed backwards only if all FF's at its output have the same logic values, the shared cost at the outputs before retiming is also one, as modeled by this term. The bound $r(u) \leq J_u$, ensures that no set of FF's, with shared cost greater than one, is ever retimed backwards across gate u.

- The second term $\rho_u \leq 0$ 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 moved across the fanout junction of gate u by forward retiming. Since a negative value of ρ_u denotes forward retiming, it reflects a cost saving in the objective function.

- 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. 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.

4.3.2 Additional Constraints

The number of FF's between gate u and its fanout v is given by $w_r(e_{uv}) = w(e_{uv}) + r(v) - r(u)$. The cost of the FF's between u and v is given by $\sum_{k=1}^{J_v + w(e_{uv})} \alpha_{class(T_u(v,k))}$, out of which $r(u)$ FF's are removed by backward retiming across gate u and $-\rho_u$ FF's are removed by forward (negative) retiming across the fanouts. The conditional sharing of FF's is automatically modeled by the sharing of the α variables amongst the fanouts. Since the cost of FF's should be same as the number of actual FF's, we get

$$w(e_{uv}) + r(v) - r(u) = \sum_{k=1}^{J_v + w(e_{uv})} \alpha_{class(T_u(v,k))} \quad - r(u) + \rho_u \ \ \forall\ v \in FO(u) \tag{9}$$

which can be rewritten as

$$w(e_{uv}) + r(v) = \rho_u + \sum_{k=1}^{J_v + w(e_{uv})} \alpha_{class(T_u(v,k))} \ \ \forall\ v \in FO(u) \tag{10}$$

Since the right hand side of Equation (10) is being minimized in the objective function, we can relax the equality to the following inequality

$$w(e_{uv}) + r(v) \leq \rho_u + \sum_{k=1}^{J_v + w(e_{uv})} \alpha_{class(T_u(v,k))} \ \ \forall\ v \in FO(u) \tag{11}$$

4.3.3 The MILP Nature of the Problem

The problem of retiming under this new FF sharing model will now be shown to be solvable using an MILP solver. Although the problem is an integer linear program (ILP), with integer values required for the r, ρ and α variables, the structure of the problem implies that both r and ρ must always be integers at any basic solution of the linear program. Therefore it suffices to set the α variables to be integers limited to the values 0 and 1, and this will result in guaranteed integer values for r and ρ.

Consider the constraints of the LP (6). To show our result, we will first summarize the well-known proof that shows that the optimal r values must correspond to integers. Due to the fact that all constraints of this LP are difference constraints, the constraint matrix has the totally unimodular property [21]. This implies that any square submatrix must have a determinat of ± 1. As a result, since the right hand side constraint vector is integral, it can be shown [21] by applying Cramer's rule that all vertices of the constraining polytope must have integer values. Consequently, regardless of the coefficients of the linear objective function, the solution must result in integer values of r, and therefore the problem can be solved as an LP rather than an ILP.

The new constraints added by us, of the form of (11), can be written as difference constraints between the r and ρ variables by temporarily relegating the α variables to the right hand side. In this situation, the left hand side of the constraint matrix remains a totally unimodular matrix. The right hand side will always evaluate to an integer if the α variables are forced to be integers. Consequently, at any legal value of α, the vertices of the constraining polytope must correspond to integer values of the r and ρ variables, and therefore any optimal solution to the problem, regardless of the coefficients of the linear objective function must have integer values of the r and ρ variables.

As a result of this, we may write the problem as a 0/1-MILP, with the α variables constrained to be 0 or 1, and no integerization constraints on the r or ρ variables, and we will be guaranteed to obtain a legal retiming solution.

4.3.4 An Example

Consider the circuit with the sharing classes in its table of logic values, as shown in Figure 9. The MILP for this circuit is

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

$$\text{subject to} \quad r(b) \leq \alpha_1 + \alpha_2 + \alpha_3 + \rho_a$$

$$r(c) \leq \alpha_1 + \alpha_4 + \rho_a$$

$$r(d) \leq \alpha_5 + \alpha_6 + \rho_a$$

$$\alpha_1 \geq \alpha_2 \geq \alpha_3$$

$$\alpha_1 \geq \alpha_4 \; ; \; \alpha_5 \geq \alpha_6$$

$$\rho_a \leq 0 \; ; \; \alpha_i \in \{0, 1\} \;\; \forall i$$

Backward 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 10(a) to the retimed circuit in Figure 10(b).

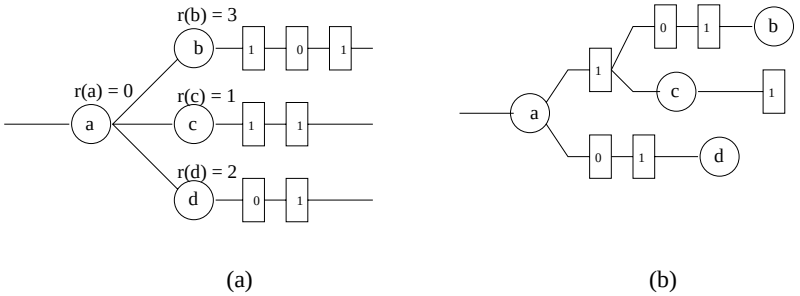

Figure 10: Example of positive retiming

Forward 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 11(a) to the retimed circuit in Figure 11(b). As can be seen one FF is shared for the edges e_{ac} and e_{ad} even though they were 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 class which are defined for backward movements. Thus these FF's can be shared and our formulation correctly models the cost.

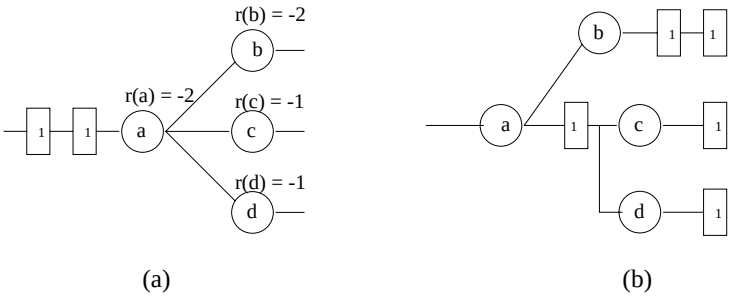

Figure 11: Example of negative retiming

5 Experimental Results

We have implemented an initial state minarea retiming based on the presentation in this work. Since obtaining an optimal solution requires complete exploration of the problem, it implies generating all the possible justification sets Δ_i's, and solving the corresponding LP's. The clock period was chosen to be the minimum clock period obtained by applying the minperiod retiming algorithm in [17]. Since enumerating the elements of an exponential-size set can take an exponential number of steps, we implement a justification algorithm that makes random choices whenever there is a non-unique justification. The LP is then solved for the corresponding Λ_i. If the lower bound , is not achieved, then we perform another random decision based justification. This process is repeated until either the lower bound is reached or a user-specified number of iterations are performed, and the best solution found is reported. Although it may seem arbitrary to use random decisions, our experimental results show that the algorithm gives us good engineering solutions that

Table 1: Minarea Initial State Retiming

Circuit	$\|G\|$	P	,	A		B		C		D	
				# FF's	T_{exec}	# FF's	T_{exec}	# FF's	T_{exec}	# FF's	T_{exec}
s27	11	6.0	3	3	0.01s	3	0.00s	3	0.01s	3	0.00s
s208.1	105	10.0	8	8	0.02s	8	0.02s	8	0.03s	8	0.03
s298	120	6.0	22	22	0.40s	22	0.44s	22	0.44s	22	0.44s
s382	159	7.0	23	23	2.59s	23	4.34s	23	4.35s	23	4.45s
s386	169	11.0	6	6	0.04s	6	0.03s	6	0.04s	6	0.03s
s344	161	14.0	19	19	1.77s	19	1.79s	19	1.77s	19	1.82s
s349	162	14.0	19	19	1.62s	19	1.62s	19	1.69s	19	1.62s
s526n	195	6.0	30	30	0.95s	30	0.95s	30	2.75s	30	0.97s
s510	212	11.0	7	7	0.12s	7	0.12s	7	012s	7	0.12s
s420.1	219	12.0	17	17	0.07s	17	0.06s	17	0.07s	17	0.06s
s641	380	74.0	19	19	0.11s	19	0.43s	19	0.44s	19	0.43s
s713	394	74	19	19	0.18s	19	0.68s	19	0.68s	19	0.69s
s967	395	12.0	35	35	28.52s	35	27.21s	35	28.05s	35	27.27s
s938	447	16.0	33	33	1.45s	33	1.53s	33	1.49s	33	1.53s
s1196	530	24.0	18	18	0.08s	18	0.07s	18	0.08s	18	0.17s
s1238	5.09	22.0	18	18	0.08s	18	0.07s	18	0.56s	18	0.08s
s1423	658	53.0	76	76	8.77s	76	9.23s	76	8.89s	76	9.31s
s1488	654	16.0	7	7	0.11s	7	0.11s	7	0.12s	7	0.11s
s1494	648	16.0	7	7	0.13s	7	0.12s	7	0.13s	7	0.12s
s3330	1790	14.0	110	110	0.58s	110	0.56s	110	0.59s	110	0.56s
s5378	2780	21.0	173	173	3m 18s	173	3m 19s	173	3m 18s	173	3m 17s
s9234.1	3271	38.0	134	134	21m 18s	134	21m 19s	134	23m 47s	134	21m 15s
s635	287	66.0	35	42	22.6s	42	22.38s	35	1.08s	39	22.5s
s953	396	13.0	27	32	32m 02s	32	27m 35s	32	31m 30s	32	27m 2s
s1269	570	19.0	84	84	0.26s	85	1m 33s	85	1m 31s	85	1m 4s
s1512	781	23.0	70	71	1h 51m 19s	72	1h 52s 1s	72	1h 56m 23s	70	1m 38s
s3271	1573	15.0	168	169	16m 46s	173	16m 5s	170	16m 40s	173	16m 29s
prolog	1602	13.0	122	124	16m 40s	125	16m 42s	125	16m 39s	125	16m 29s
s3384	1686	27.0	167	168	55m 42s	169	1h3m 18s	169	1h 2m 56s	169	51m 3s
s15850.1	9618	63.0	525	544	3h 9m 56s	540	4h 2m 36s	542	3h 57m 7s	544	3h 59m 5s

are close to the (possibly unachievable) lower bound. Other possible stopping criteria could be (a) having the best result obtained so far be within a given percentage of the lower bound, or (b) obtaining no improvements in the best solution for a given number of iterations, etc. If there are no gates with conflicts, then the LP is the dual of a network flow mincost flow problem, and is solved using a network simplex algorithm [16]. If, however, we have to solve the general MILP we use the public domain MILP solver, *lp_solve* [22].

We present results on the ISCAS89 [23] benchmark suite in Table 1. For each circuit, we show the number of gates $|G|$, the target clock period $|P|$, and the lower bound on the number of FF's obtained by [16] , . We also show the minimum number of FF's obtained with equivalent initial state and the execution time T_{exec} for all the tasks including solving the LP for all iteration on a HP 9000/777 C110 workstation. In the absence of initial state values for the benchmark circuits we present results for four different initial state assignment. Case A has all FF's initialized to 0, while case B has all initialized to 1, case C and D are for random state assignments. As can be seen from the results, for many circuits the lower bound is achieved in a small number of iterations for almost any initial state. In fact in almost all of these cases the lower bound , is obtained in the first iteration itself. For some circuits the lower bound was not reached; these are shown in the lower part of the table. This, however, does not imply that the solution obtained is not optimal since

the lower bound is not always achievable with equivalent initial state. For these circuits, we report the best solution obtained in 50 (5 for s15850.1) iterations.

We observed that in all of our test circuits, the number of gates requiring the 0-1 MILP model was less than 2% of the total number of gates. In fact, in several of them, the number of gates that required the new MILP model for register sharing was very small (less than 10). This kept the number of integer variables well within the acceptable range for all our circuits, making our approach practical.

In the circuits where the lower bound , is not achieved the solution reported by our algorithm is very close to , , and therefore, corresponds to a good engineering solution. Since the optimal number of FF's in a circuit depend on the initial state of the original circuit, some variation in the number of FF's and execution time is obtained for different initial states. For `s635, s1269 and s1512` the lower bound was seen to be achieved for only some initial states.

6 Conclusion

We have presented a method to obtain minarea retiming of control logic subject to a given target clock period and an equivalent initial state. Any minarea retiming algorithm, that does not consider initial states will, in general, not give a solution with a valid equivalent state and hence cannot be used for control logic, where initial states are important. Our method, on the other hand, will always result in a retimed circuit with an equivalent initial state, i.e., the retimed circuit starting in the equivalent initial state will have the same behavior as the original circuit starting in its given initial state. Unlike conventional minarea retiming algorithms our approach can be used for performance constrained, area optimization of control circuits. This approach also has applications in minarea retiming of circuits that contain different types of memory elements that can not be shared with each other, e.g. load enable registers.

We provide a simple way to incorporate the constraints for ensuring that the resulting retiming has an equivalent initial state. This is achieved by imposing upper bounds on the retiming variables so that any retiming respecting those bounds will have an equivalent initial state. This equivalent state can easily be found after the retiming by using the information stored from the justification phase. The technique also utilizes a new approach that incorporates conditional FF sharing since the idea of mirror vertices used by Leiserson and Saxe to model unconditional FF sharing [2] cannot be extended to the initial state retiming problem. The solution approach searches the justification space for the initial states and for each possible justification, solves an LP. The exploration of the justification space can be stopped by the user at any time, and it was seen that for all circuits tested, good engineering solutions that were close to a (possibly unachievable) lower bound were found by the technique after a small amount of exploration.

The work in [24] showed that backward retiming with equivalent initial states such as the one in Figure 1 can always be obtained if the reset signal is expressed explicitly. This however requires the addition of a multiplexor before the FF and thus changes the path delays in the circuit. This may cause the clock period of the circuit to increase and is, therefore, not considered here.

References

[1] C. Leiserson, F. Rose, and J. B. Saxe, "Optimizing synchronous circuitry by retiming," in *Proceedings*

of the 3rd Caltech Conference on VLSI, pp. 87–116, 1983.

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

[3] K. A. Sakallah, T. N. Mudge, and O. A. Olukotun, "Analysis and design of latch-controlled synchronous digital circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 11, pp. 322–333, Mar. 1992.

[4] B. Lockyear and C. Ebeling, "Optimal retiming of level-clocked circuits using symmetric clock schedules," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, pp. 1097–1109, Sept. 1994.

[5] K. N. Lalgudi and M. Papaefthymiou, "DELAY: An efficient tool for retiming with realistic delay modeling," in *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 304–309, 1995.

[6] T. Soyata, E. G. Friedman, and J. H. Mulligan, Jr., "Incorporating internconnect, register and clock distribution delays into the retiming process," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, pp. 165–120, Jan. 1997.

[7] J. Monteiro, S. Devadas, and A. Ghosh, "Retiming sequential circuits for low power," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 398–402, 1993.

[8] K. N. Lalgudi and M. Papaefthymiou, "Fixed-phase retiming for low power," in *Proceedings of the International Symposium of Low Power Electronics and Design*, pp. 259–264, 1996.

[9] S. Malik, E. M. Sentovich, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Retiming and resynthesis: Optimizing sequential networks with combinational techniques," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 10, pp. 74–84, Jan. 1991.

[10] G. De Micheli, "Synchronous logic synthesis: Algorithms for cycle time minimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 10, pp. 63–73, Jan. 1991.

[11] V. Singhal, C. Pixley, R. L. Rudell, and R. K. Brayton, "The validity of retiming sequential circuits," in *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 316–321, 1995.

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

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

[14] I. Y. Spillinger, L. Stok, and G. Even, "Improving initialization through reversed retiming," in *Proceedings of the European Design and Test Conference*, pp. 150–154, 1995.

[15] N. Shenoy and R. Rudell, "Efficient implementation of retiming," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 226–233, 1994.

[16] N. Maheshwari and S. S. Sapatnekar, "An improved algorithm for minimum-area retiming," in *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 2–7, 1997.

[17] S. S. Sapatnekar and R. B. Deokar, "Utilizing the retiming skew equivalence in a practical algorithm for retiming large circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, pp. 1237–1248, Oct. 1996.

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

[19] S. Kundu, L. Huisman, I. Nair, V. Iyengar, and L. Reddy, "A small test generator for large designs," in *Proceedings of the IEEE International Test Conference*, pp. 30–40, 1992.

[20] J. B. Saxe, *Decomposable Searching Problems and Circuit Optimization by Retiming: Two Studies in General Transformations of Computational Structures*. PhD thesis, Carnegie-Mellon University, Pittsburgh, PA, 1985.

[21] M. S. Bazaraa, J. J. Javis, and H. Sherali, *Linear Programming and Network Flows*. New York, NY: John Wiley, 1977.

[22] M. Berkelaar, *LP_SOLVE User's Manual*. Eindhoven University of Technology, Eindhoven, The Netherlands, June 1992.

[23] F. Brglez, D.Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 1929–1934, 1989.

[24] V. Singhal, S. Malik, and R. K. Brayton, "The case for retiming with explicit reset circuitry," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 618–625, 1996.