The Case for Retiming with Explicit Reset Circuitry

Vigyan Singhal
Cadence Berkeley Labs
Berkeley, CA 94704
`vigyan@cadence.com`

Sharad Malik
Princeton University
Princeton, NJ 08544
`malik@princeton.edu`

Robert K. Brayton
University of California at Berkeley
Berkeley, CA 94720
`brayton@eecs.berkeley.edu`

Abstract

Retiming is often used to optimize synchronous sequential circuits for area or delay or both. If the latches[1] that are retimed have a hardware reset value, the initial state of the circuit must also be retimed, i.e. an initial state must be derived for the retimed circuit. Previously, it has been suggested that this can be avoided if the hardware reset signals are represented explicitly. However, it was thought that this adds unnecessary area and restricts the space of possible retimings. In this paper we demonstrate that this is not the case. In addition, we show that this methodology does not require the restriction that all reset signals be asserted at the beginning of circuit operation— a restriction that was imposed by existing algorithms for determining the retimed initial state. Finally we show how our explicit reset (ER) framework enables us to retime when some latches may be driven by different hardware resets, and some others may not have any hardware resets. We also consider the case where the resets are asynchronous. We expect these solutions to the "retimed initial state" problem to help increase the practical applicability of retiming.

1 Introduction

This research addresses the optimization of synchronous digital circuits. A synchronous circuit is defined informally as an interconnection of combinational logic gates (**gates**) and synchronizing memory elements (**latches**), where each cycle contains at least one latch. Retiming is a well known optimization technique – it moves latches across combinational logic in order to minimize the delay and/or the area of the circuit. Recently, efficient retiming algorithms have been proposed for large industrial-sized circuits [9] which take this technique one step closer to wider adoption in practice. However, one nagging problem with using retiming in practice is retiming latches which have hardware reset values (alternatively called initial values) [3]. In practice, most designs have a few such latches (called **reset latches** in this paper), and the remaining latches do not have any hardware reset (called **no-reset latches**).

For the purposes of synthesis and analysis it is often convenient to represent each reset latch with a no-reset latch and explicit hardware circuitry. This framework (which we call the **explicit reset** (ER) framework) was proposed by [1, 7]. Such a representation also provides a natural solution for the problem of retiming the initial state, since now no latch has a known initial value. Also, it makes it possible to reason about the behavior of circuits which have several different hardware reset lines without making an implicit assumption that (a) the circuit operation starts only after *all* these reset lines are asserted in the same cycle, and (b) the reset lines are never asserted after this first cycle of operation. We call this assumption the **global reset activation** (GRA) assumption.

[1] We use the term *latches* here to refer to memory elements. The actual implementation may be as either edge-triggered or level sensitive devices.

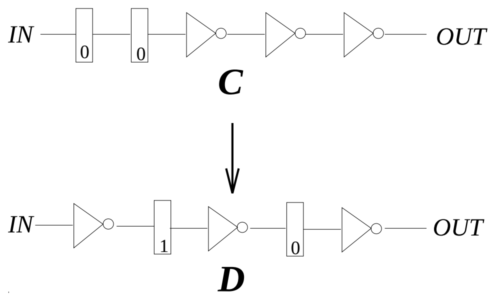

Figure 1: Retiming reset latches and the reset state

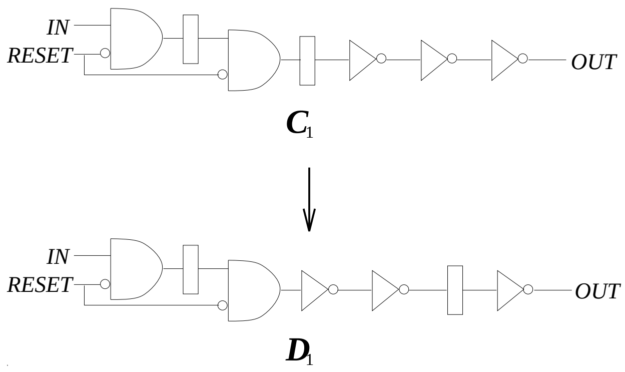

Figure 2: Retiming after the reset mapping

Relaxing this assumption is important because frequently we cannot enforce these two requirements on every hardware reset signal since reset signals may themselves be driven by other logic in the circuit [2].

However, it has been suggested that the explicit representation of reset circuitry restricts the retiming moves and adds unnecessary overhead [7, 8, 11]. Consider, for example, the circuit C in Figure 1. The rectangular boxes denote latches. Suppose each inverter has a delay of d and the reset value of each latch is 0 (shown inside the latch in the figure). This circuit can be retimed to obtain circuit D which has a total delay of d, as opposed to the delay of $3d$ of the original circuit (notice that we have **retimed the reset state** also so that the new reset values are 1 and 0, respectively). However, suppose we represent the reset circuitry explicitly to obtain the circuit C_1 from our original circuit C by using the **reset mapping** (replacing reset latches with no-reset latches plus reset circuitry) as shown in Figure 2. This circuit has an extra input, the global

[2] This is why we use the phrase "reset state" instead of "initial state" in the remainder of the paper.

reset line. The additional combinational logic added in the circuit (the two AND-gates) has delay 0. If we retime this circuit for minimum delay, it seems that we can only retime the front latch forward across the inverters; we can no longer retime the rear latch because the AND gate is in the way. Thus, it looks like we can no longer achieve a minimum delay of d, as before. So, it appears that the reset mapping may prevent a synthesis tool to reach regions of the design space which were reachable before this mapping.

In this paper, we first show that the above situation is rectifiable, and all retimings which can be obtained under the implicit reset (IR) framework (where the reset mapping is not used) can equivalently be obtained in the ER framework, and *without any additional area overhead*. This rectification is made possible with the use of a resynthesis step along with the retiming step (a very special case of the general class of transformations proposed in [7, 2]). We also show that a retiming of the reset state across primary outputs and primary inputs (OI), which is used during a retiming of the reset state with a global STG-based analysis of the whole circuit [11] can be accomplished in the ER framework. We show the correctness of both the above transformations (retiming across gates and across OI) *without making the GRA assumption* which was needed for the global STG-based analysis [11].

We then extend our results to show how the ER framework enables the retiming of a set of latches, some of which may have different hardware reset lines and some may not have any reset line. These generalizations provide a comprehensive solution to the "retimed reset state" problem.

This paper discusses ways of retiming the reset state and establishing the correctness of these transformations *for a given retiming*. Thus, this is orthogonal to the work done by Even *et al.* [3] which, given a target delay, attempts to minimize the extra logic needed to achieve the retiming.

2 Preliminaries

2.1 Leiserson-Saxe Retiming Graph

Leiserson and Saxe introduced retiming [6] using a graph-theoretic model. A design is modeled as a finite edge-weighted directed graph $G = (V, E)$. Each vertex in V represents either a gate in the design, or a special dummy node called the *host*. There is an edge in E from gate g_1 to gate g_2 if an output of g_1 is an input to g_2. There is an edge from the host to each gate fed by a primary input and there is an edge from each gate which feeds a primary output to the host. The non-negative weight of an edge represents the number of latches on the corresponding path in the design. A retiming of a design is an assignment of an integer $lag(v)$ to each vertex v which denotes the number of latches moved across the gate corresponding to the vertex. If the lag is positive (negative), the latches are removed from each fanout (fanin) edge of the gate and placed at each fanin (fanout) edge of the gate. Thus, for edge (u, v) with weight w, the weight after retiming is $w + lag(v) - lag(u)$. As an example, consider the retiming graphs showing the lag assignment (in Figure 3) for the circuit retiming shown later in Figure 8 (vertices H, 1, 2, and 3 represent the host vertex, the AND-gate, the inverter, the fanout junction, respectively). The retiming corresponds to the lag assignment shown above each vertex. An alternative lag assignment which gives the same circuit is given below each vertex. Notice that the upper assignment achieves this by backward retiming moves across the junction and the inverter while the lower assignment retimes forward across the output-input (OI) and the AND-gate. Thus, different retimings (lag assignments) can result in the same circuit.

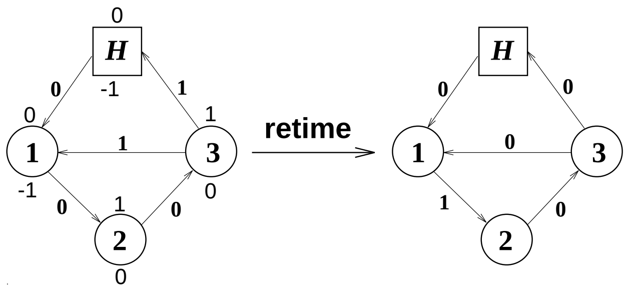

Figure 3: Retiming graph for the retiming in Figure 8

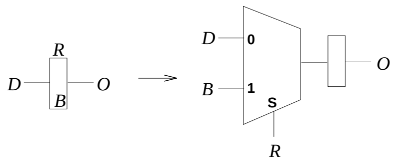

Figure 4: The reset mapping

2.2 The ER framework

The ER framework replaces reset latches by no-reset latches plus some reset circuitry. This replacement is possible for only synchronous reset latches [3]. A synchronous reset latch with reset signal R (shown above the latch in the figure), reset value B and data input D is represented by a no-reset latch and a multiplexor (MUX) driving the no-reset latch (Figure 4). We denote this MUX-gate by MUX(R, D, B). The MUX can be simplified to an AND-gate (an OR-gate) if B is 0 (1). We call this transformation the **reset mapping** and denote it by Φ. This is useful for analyzing the behavior of designs and of transformations on these designs without giving a special status to the reset line (e.g., the GRA assumption). An inverse transformation Φ^{-1} can always undo the reset mapping [4].

2.3 Generalizations of Circuit Model

We consider three primary generalizations of the traditional model for retiming the reset state. These three generalizations are depicted as the three dimensions in Figure 5:

[3]Unless specified explicitly, we assume synchronous reset latches. We deal with the asynchronous case in Sections 3.3 and 3.5.

[4]This may be desirable because for a given technology library, it may be cheaper to implement a circuit with reset latches instead of explicit reset circuitry.

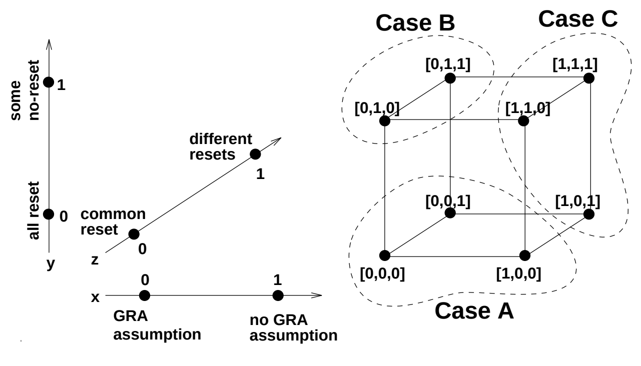

Figure 5: Generalization of circuit model

1. With or without the GRA assumption (x-axis). Not requiring the GRA assumption means that the reset lines need not be asserted at the start of the operations of the design, and that the reset lines may be asserted anytime during the circuit operation.

2. All reset latches versus some no-reset latches (y-axis). The former requires all latches to be reset latches, the latter allows no-reset latches.

3. Single reset signal versus multiple reset signal (z-axis). The former requires a single signal to drive all reset latches, the latter permits different reset latches to be driven by different signals.

These three possible generalizations extend the traditional model, which lies at [0,0,0] in Figure 5, to 8 possibilities (corresponding to the 8 vertices of the unit cube). The unit cube defines an obvious partial order over the possible models (for example, the model at [0,1,1] is more general than the model at [0,0,0]). If we give a method for retiming reset states for a more general model, it will be applicable to a less general model, but not vice-versa.

The generalizations are not totally independent. If we make the GRA assumption, the models for single versus multiple reset lines are indistinguishable— if we require all reset lines to be asserted exactly and only before the first clock cycle, they might as well be the same reset line. Thus points [0,0,0] and [0,0,1] are equivalent, and so are [0,1,0] and [0,1,1]. We will not consider [0,0,1] and [0,1,1] any more.

Our primary focus will be the models on the x-axis (corresponding to points [0,0,0] and [1,0,0]). We show that not only can we achieve all retimings of the reset state in the ER model, we can do these without making the GRA assumption. Section 3.1 covers this case (Case A). Next, we show that it is possible to retime the reset state for the model at [0,1,0] (GRA assumption but some latches may be no-reset latches). Section 3.2 covers this case (Case B). In Section 3.4 we show that if we do not make the GRA assumption, all these retiming moves may not be possible when we also have no-reset latches. We then show which retimings of the reset state are possible (this solution is applicable for the model at [1,1,1], and thus for the models at [1,1,0] and [1,0,1]); for the remaining retimings, the reset circuitry has to separated from a latch in order to retime the latch. This is described as Case C.

The analysis discussed above holds for when the reset on the latches is synchronous, i.e. if hardware reset is asserted at the clock edge, the reset value is activated at the next clock cycle synchronously. On the other hand, for asynchronous resets, the desired reset value can be locked in if the hardware reset is asserted at any time (and not necessarily at the clock edge). We will argue that for the asynchronous case, the solutions for Cases A and B are applicable (Section 3.3). However, the solution for Case C is more restrictive than that for the synchronous case; we discuss this in Section 3.5.

Throughout this paper we will assume that the clock periods are long enough to allow all logic to evaluate to a steady state. This means that for asynchronous resets, the time between a positive assertion of the reset line and the next clock edge is sufficient for the next-state or output logic following the latch to complete its evaluation.

2.4 Criterion for Validity of Transformations

For the model at [0,0,0], two circuits are equivalent if their respective reset states are equivalent (equal output sequences for each input sequence). However, in general, we need a stronger condition which does not require special assumptions about the resettability of the design.

Leiserson and Saxe [6] show the validity of retiming transformations under the following condition for replacement:

Definition 1 *A design D is a* **sufficiently old replacement** *of design C if there exists a non-negative integer p such that for every state d that D can be in q cycles after power-up for any $q > p$, there exists a state c in C such that d and c are equivalent.*

Notice that at the power-up of a design, each latch non-deterministically assumes a Boolean value (under the GRA assumption, only the no-reset latches get arbitrary values; the reset latches assume their designated reset values). If n in the above condition is small enough, we can usually afford to wait these extra clock cycles before beginning circuit operation. This notion of replacement is very similar to (but stronger than) the notions of n-delay replacement [10] and c-cycle redundancy [5]. To save space, henceforth, we say that D is **valid replacement** of C if and only if D is a sufficiently old replacement of design C. The notion of valid replacement is clearly transitive, and we will make use of this transitivity later. Note that this notion does *not* require a special status of the reset lines, specifically that the reset lines have to be asserted at the beginning of circuit operation.

If design D is obtained from design C by a sequence of retiming moves, where only no-reset latches are retimed (and not the reset circuitry), then:

Theorem 1 ([6]) *D is a valid replacement of C.*

3 Retiming Reset Circuitry

Retiming the reset state is accomplished in our ER framework by moving the reset circuitry simultaneously with the latches during retiming. We refer to this operation as *retiming the reset circuitry*. First we show that all retimings of the reset state in the IR framework can be achieved in the ER framework, and moreover, we can generalize the circuit model so that we no longer make the GRA assumption.

3.1 Case A: single reset line and no no-reset latches

Here we are interested in the models at [0,0,0] and [1,0,0] (Figure 5). The traditional solution retimes the implicit reset state (in the IR framework) and works for the model at [0,0,0], i.e. it makes the GRA assumption. Since we will relax the GRA assumption, we will need to prove the validity of the transformation as specified in Section 2.4.

The standard strategy to retime the reset state for individual forward or backward retiming moves (as described in [1]) is to take the forward or backward functional image. We show how this can be achieved in the ER framework without making the GRA assumption in Section 3.1.1. However, this strategy, by virtue of being local, can make local decisions which may need subsequent backtracking. To avoid this, it may be possible to determine the final value of the reset state by looking at the entire set of retiming moves and analyzing their effect on the transition graph of the design [11]. In Section 3.1.2 we argue the correctness of this under the ER framework without making the GRA assumption.

3.1.1 Local retiming of the reset state

Consider retiming across a multi-output gate $F = (f_1, \ldots, f_n)$, where each f_i is a function from input space $\{0,1\}^m$ to $\{0,1\}$ (junctions are modeled as single input multi-output gates).

Bartlett *et al.* [1] describe the standard method for obtaining the new reset state if we retime reset latches across the gate F.

Suppose the original design has one reset latch at each of the inputs of the gate, with reset values $\vec{a} = (a_1, \ldots, a_m)$, and we want to retime the latches forward across F. The reset values on the reset latches in the retimed design are simply given by $\vec{b} = (b_1, \ldots, b_n)$, where each $b_i = f_i(\vec{a})$. Now, consider backward retiming across F. Let the reset values of the latches on the outputs of F be $\vec{b} = (b_1, \ldots, b_n)$. If there is a set of values $\vec{a} = (a_1, \ldots, a_m)$ such that for each $i \in \{1, \ldots, n\}$: $b_i = f_i(\vec{a})$, we set the reset values on the retimed latches to $\vec{a}$. If there is no such vector, the reset state cannot be retimed backward; we discuss this scenario later in this section.

Now, we give a procedure of retiming in the ER framework i.e. all latches are no-reset latches along with the reset circuitry. This procedure is such that if circuit D is obtained from C (where all latches are reset latches) after one atomic retiming move, as described by the above re-mapping of the reset state, then the mapped circuit $C_1 = \Phi(C)$ can be retimed to D_1 such that $D_1 = \Phi(D)$. We show that the retimed circuit D_1 (or D) is a valid replacement of the original circuit C_1 (or C).

Proposition 2 *For each retiming of latches and reset state in a circuit with reset latches, we can achieve the same transformation in the ER framework such that the retimed design is a valid replacement of the original design in the [1,0,0] model.*

Proof: Suppose we are retiming across a multi-output gate with the functionality $F = (f_1(x_1, \ldots, x_m), \ldots, f_n(x_1, \ldots, x_m))$.

There are two cases:

a) The move from C to D is a forward move. Refer to Figure 6 (for the IR framework, the common hardware reset line is omitted from circuit figures). Circuit C_1 represents $\Phi(C)$. Thus if in circuit C the i-th latch has the reset value a_i, then in circuit C_1 the i-th latch is represented with a no-reset latch and MUX(R, x_i, a_i). The retiming from C_1 to D_1 consists of two steps. In Step 1, just the latches are retimed. In Step 2, the reset circuitry is resynthesized across the element. Let $\mathcal{F}(R, x_1, , x_2, \ldots, x_m, a_1, a_2, \ldots, a_m)$ be the multi-output function being computed by the combinational part of this circuit. This resynthesis follows directly from the Shannon decomposition of $\mathcal{F}$ about R.

$$\begin{aligned}\mathcal{F} \quad = \quad & R \cdot \mathcal{F}(R = 1, x_1, x_2, \ldots, x_m, a_1, a_2, \ldots, a_m) + \\ & \overline{R} \cdot \mathcal{F}(R = 0, x_1, x_2, \ldots, x_m, a_1, a_2, \ldots, a_m)\end{aligned}$$

The RHS simplifies to:

$R \cdot F(a_1, a_2, \ldots, a_m) + \overline{R} \cdot F(x_1, x_2, \ldots, x_m)$

This is equivalent to the circuit with multiplexors at the output of F with $b_i = f_i(a_1, \ldots, a_m)$. Step 1 causes valid replacement (Theorem 1). Step 2 is a valid combinational replacement, and by the transitivity of valid replacements, D_1 is a valid replacement of C_1. Now we can map the no-reset latches plus reset circuitry in D_1 to reset latches with the appropriate reset values (using Φ^{-1}) to get the desired retimed design D.

b) The move from C to D is a backward move. The result follows by reversing the steps of case (a). Since the retiming can be achieved for reset latches, there must exist values $(a_1, \ldots, a_m)$ such that for each $i \in \{1, \ldots, n\}$: $f_i(a_1, \ldots, a_m) = b_i$. ■

Thus, when backward or forward retiming of reset latches across combinational logic elements is possible by re-mapping the reset values of reset latches, this can always be achieved in the ER framework. Notice, that in essence, we have performed a special case of the general retiming and resynthesis paradigm [7]. However, we like to think of the step of resynthesizing the reset circuitry across the logic element as *retiming* the reset circuitry. For an example, consider the retiming of Figures 1 and 2. Using the procedure described above, we can now obtain the desired retiming with delay d, as shown in Figure 7.

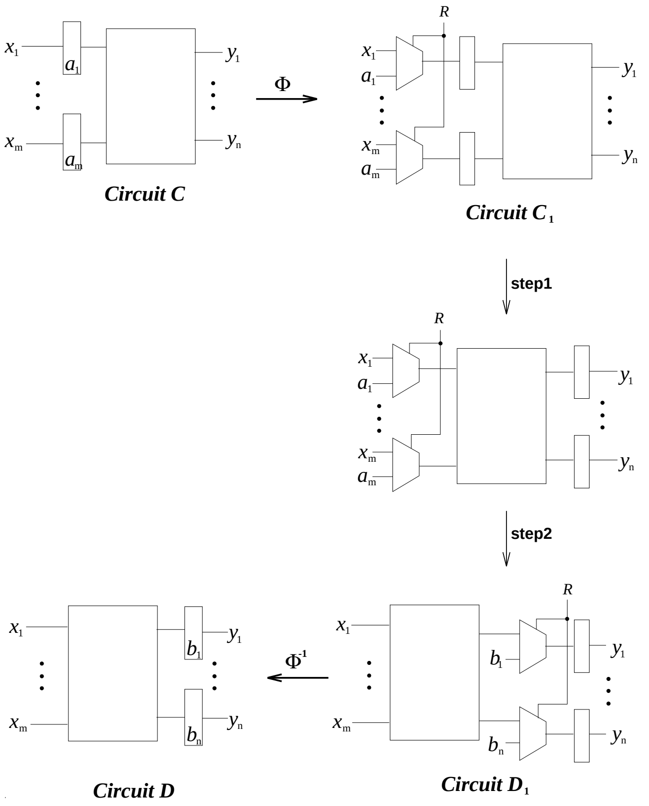

Figure 6: Forward retiming of the reset circuitry

Separating reset circuitry for backward retiming

In case (b) of Proposition 2 we saw that for retiming the reset state backwards, we need the existence of the backward image of the reset state across F.

Sometimes it may be impossible to retime reset latches without adding extra logic in the design. This is illustrated by the following example (due to Touati and Brayton [11]). Consider the upper circuit in Figure 8 where both latches are reset latches driven by the same reset line. If we want to retime these latches (and the reset value, denoted by "?") backward across junction and the NOT-gate, to obtain the circuit drawn below, we would like to determine an equivalent reset state for this circuit. The goal in [11] is to determine a state such that the reset state of the lower circuit is equivalent to the reset state of the upper circuit (so that when the reset line is pulled to 1, both circuits have equivalent behavior). However, it is clear from the STG of the lower circuit that no such state exists. The only way to achieve a design equivalent to the above design is by adding extra logic to the circuit. For such designs, Touati and Brayton give an algorithm, based on an analysis of the original STG, which adds redundant combinational logic so that after retiming, the new STG does have a state equivalent to the original reset state.

The ER framework provides an alternate and natural way of determining this extra logic. The basic idea is to do the reset

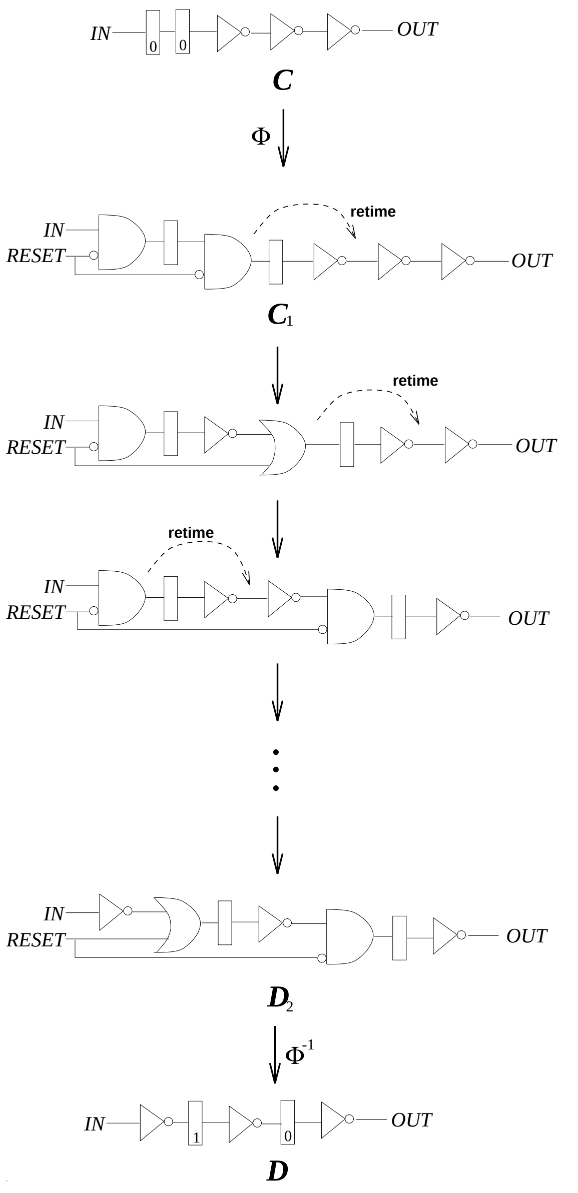

Figure 7: Example of retiming reset circuitry.

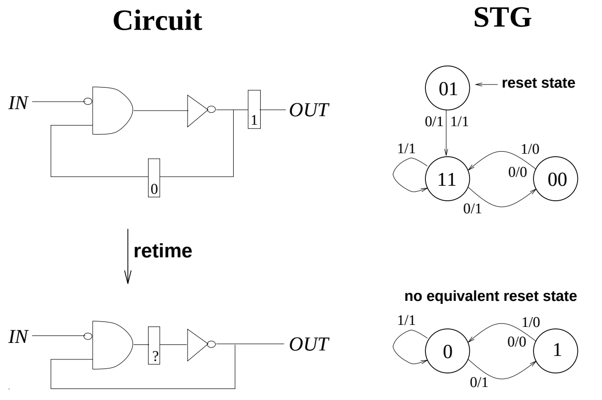

Figure 8: Example of impossible retiming without adding logic

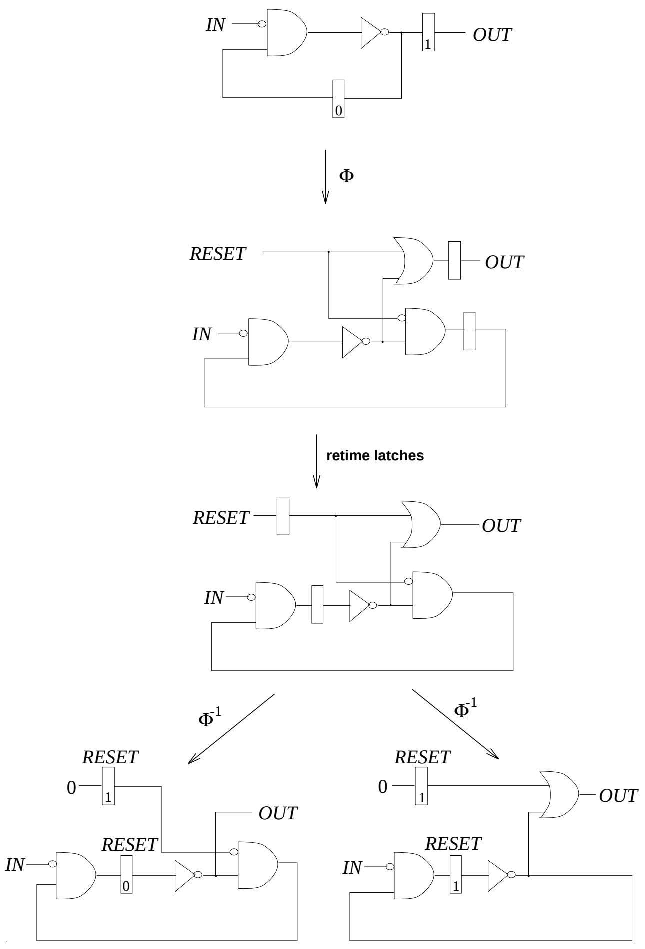

Figure 9: Separating the reset circuitry from no-reset latches.

mapping, and if we are unable to locally retime the reset circuitry along with the no-reset latch (this happens for a backward retiming move as we saw in Section 3.1.1), leave the reset circuitry in place and just retime the no-reset latch. For the example in Figure 8, we can achieve the desired retiming as shown in Figure 9 (notice that in this example, the fanout junction, which we model as a 1-input 2-output gate, does not have a backward image of the reset value 10). In order to get back to a model with all reset latches, we have the option to apply a Φ^{-1} mapping step at the end; as we see in this example, we may have a choice and it make sense to make the choice which facilitates more future retimings of the reset circuitry.

3.1.2 Global retiming of the reset state

The procedure for obtaining the new reset values for each atomic retiming move, described at the beginning of the previous section, can run into problems because it uses only local knowledge when making decisions about which new reset value to choose for a backward retiming move. Because of reconvergence in the logic, one choice may cause a conflict in retiming the reset state in the future, whereas another choice may not. This may result in backtracking to determine a value consistent with the different reconvergent paths.

An elegant solution to avoid this is provided in [11] where the global behavior of all latches is analyzed, and the value of the reset state for the entire sequence of retiming steps is found directly. The solution requires analysis over the implicit state transition graph of the design; a solution exists if there exists a state which goes to the designated reset state of the original design after k transitions, where k equals $\max_v\{lag(v) - lag(host)\}$ over each vertex v of the retiming graph. Touati and Brayton proved that their algorithm obtains the retiming of the reset state such that the new designated reset state is equivalent to the original reset state.

While reset state equivalence is sufficient for the model at [0,0,0], we claim that it is also sufficient for [1,0,0]. As long as the (unique) reset line is not asserted, the retimed circuit is clearly a valid replacement for the original (since the reset circuitry does not affect circuit behavior Theorem 1 asserts the validity). When the reset line is asserted, every existing latch value is lost (since *all* latches have reset lines) and is replaced by the designated reset value. Thus, we only need state equivalence of the global reset states of the original and the retimed design to show validity. This informal argument illustrates the applicability of the Touati-Brayton algorithm for the global retiming of the reset circuitry for Case A.

Global vs local retiming

While the global retiming solution has the ability to find a solution as long it can find a state which reaches the reset state in k steps, the big drawback of this approach is that it needs to sequentially justify a given state over k previous states of the designs. This limits its applicability to small to moderately sized designs. For larger designs, the local retiming method discussed in Section 3.1.1 is the only possible approach.

We must also note that given a complete backtracking algorithm to find the "right" choices for backward reset state computations, the local retiming algorithm of Section 3.1.1 is sufficient to achieve all retimings. Using the global retiming algorithm may just a more efficient method to avoid this backtracking. Henceforth, we will not consider showing the correctness of a global retiming procedure for the remaining models.

3.2 Case B: GRA assumption with no-reset latches

Let us now consider the case where some of the latches are no-reset. Our goal is to obtain a valid retiming of the reset circuitry. It is sufficient to show how this is accomplished for the case of local retiming of the reset state. Consider Figure 6 again. However, now assume that only the first k latches are reset, the others are no-reset.

Let us consider forward retiming first. In this case, converting the no-reset latches to reset latches with arbitrary reset values is a valid operation. To see why this is so, for the environment observing the circuit, all behaviors that correspond to any power-up values for the no-reset latches values must be acceptable, i.e. the observing environment cannot expect any behavior that depends on a specific reset state for the no-reset latches. Thus we can select any arbitrary power up value. Now, the situation is exactly the same as that for the forward retiming case of Case A (in Section 3.1), and thus the retiming circuitry can be retimed forward along with the latches.

In the case of backward retiming, we just need to find the inverse image for the values in the reset latches. Let the reset values of the reset latches on the outputs of F be $\vec{b} = (b_1, \ldots, b_k)$ (only the first k are reset latches). If there is a set of values $\vec{a} = (a_1, \ldots, a_m)$ such that for each $i \in \{1, \ldots, k\}$: $b_i = f_i(\vec{a})$, we set the reset values on the retimed latches to $\vec{a}$. If there is no such vector, we cannot retime the reset circuitry; we have to separate the reset circuitry from the latch if we want to achieve the retiming. This guarantees that for the first cycle we will get the correct values on the outputs of F for the signals which had reset latches, and for the others the values will be one of the acceptable power-up values.

Note that for both the cases above we need the GRA assumption. We replaced the arbitrary power-up values of no-reset latches by specific values. This satisfies sufficiently old replaceability (Definition 1) for $n = 0$ because the new designated reset state is one of the many power-up states of the original design. The rest of proof for the validity of these transformations follows from the arguments in Section 3.1.

3.3 Asynchronous resets for Cases A and B

For the 4 models on the left plane in Figure 5, since the GRA assumption holds, the reset lines are asserted before the circuit operation. Thus for the transformations described in Sections 3.1 and 3.2, it does not matter whether the reset is synchronous or asynchronous.

We will show the validity of the retiming transformations for the model at [1,0,0]. We will consider forward and backward retiming moves across a multi-output gate $F = (f_1(x_1, \ldots, x_m), \ldots, f_n(x_1, \ldots, x_m))$. Consider circuits C and D as shown in Figure 6 (we assume that the resets on the latches are asynchronous instead of the synchronous assumption in Section 3.1). Let $\vec{a} = (a_1, \ldots, a_m)$ and $\vec{b} = (b_1, \ldots, b_n)$. We need the following result to show the validity.

Lemma 3 *If for $1 \leq j \leq m$, $b_j = f_j(\vec{a})$ then the reset states of C and D are equivalent.*

Proof: For any input sequence, we claim that the output sequences of circuits C and D starting from state $\vec{a}$ and $\vec{b}$, respectively, are identical. The outputs are identical and equal to $(f_1(\vec{a}), \ldots, f_m(\vec{a}))$ on the first cycle. Consider the k-th output vector for $k > 1$. If the reset R is asserted during the k-th cycle, the outputs of C and D are both $(f_1(\vec{a}), \ldots, f_m(\vec{a}))$. Otherwise, assume that $(k-1)$-th input vector is $(R, \vec{c})$ where $\vec{c} = (c_1, \ldots, c_m)$ (note that if R is ever 1 during the $(k-1)$-th cycle and if there is a combinational path in the environment from $\vec{y}$ to $\vec{x}$, then $\vec{c}$ may be a combinational function of $\vec{a}$ or $\vec{b}$). For this case, the k-th output vector for both C and D is $(f_1(\vec{c}), \ldots, f_m(\vec{c}))$. ■

For the asynchronous resets, the following shows the validity of the forward retiming move (taking the forward image of the reset state $\vec{a}$) and the backward retiming move, if possible (taking the backward image of $\vec{b}$):

Proposition 4 *For the model at [1,0,0], if for $1 \leq j \leq m$, $b_j = f_j(\vec{a})$, D and C are valid replacements of each other.*

Proof: We will only prove that D is a valid replacement of C; the other direction is similar. Consider any state that D can be in q cycles after power-up where $q > 1$. It is obvious that D will either be in state $\vec{b}$ or in state $(f_1(\vec{c}), \ldots, f_m(\vec{c}))$ where the inputs on the $(q-1)$-th cycle are $\vec{x} = \vec{c}$ and an arbitrary value of R. From Lemma 3, this state is equivalent to either $\vec{a}$ or $\vec{c}$ of design C. Thus D is a sufficiently old replacement of C (Definition 1). ■

For the backward retiming case, when the backward image across F is not defined, it is possible to obtain a solution similar to that for the synchronous case in Section 3.1. Consider the solution shown in Figure 10 (the X_i reset values in D are arbitrary and can be chosen suitably to facilitate future retimings).

Proposition 5 *For the model at [1,0,0], D is a valid replacement for C.*

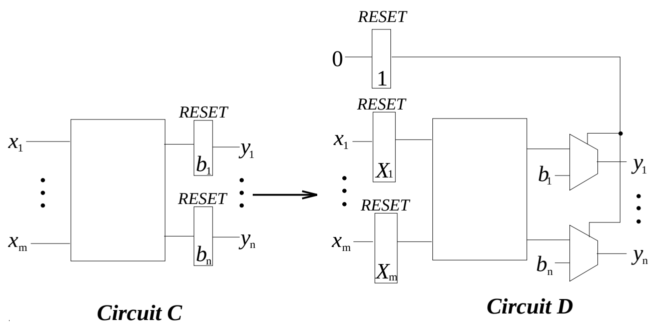

Figure 10: Backward retiming for asynchronous resets

Proof: For any arbitrary $\vec{c} = (c_1, c_2, \ldots, c_m)$ we claim that (1) state $(1, c_1, c_2, \ldots, c_m)$ of circuit D is equivalent to the state $\vec{b} = (b_1, \ldots, b_n)$ of circuit C, and (2) state $(0, c_1, c_2, \ldots, c_m)$ of circuit D is equivalent to state $(f_1(\vec{c}), \ldots, f_n(\vec{c}))$ of circuit C (the proofs are very similar to that of Lemma 3). Thus any state of circuit D is equivalent to some state of circuit C showing that D is a sufficiently old replacement of C (Definition 1). ■

3.4 Case C: most general circuit model

The analysis for Case A can be extended to the case where different latches may have different hardware reset lines. It is sufficient to show how this is accomplished for the case of local retiming of the reset state. We consider the case for forward and backward moves separately.

Forward retiming

We will consider the same sequence of steps as that for Case A as shown in Figure 6. For simplicity we assume that $m = 3$ and the latch 1 is controlled by reset R_1 and latch 2 is controlled by R_2 and latch 3 is a no-reset latch. First the reset circuitry is made explicit using the reset mapping to give C_1. Next, in Step 1 the latches are retimed forward across F. Let $\mathcal{F}$ be the combinational part of this circuit. We will first try and retime the reset circuitry for R_1 across F. We consider the Shannon expansion of $\mathcal{F}$ about R_1 and R_2:

$$\begin{aligned}\mathcal{F} &= R_1 \cdot (R_2 \cdot F(a_1, a_2, x_3) + \overline{R_2} \cdot F(a_1, x_2, x_3)) + \\ &\quad \overline{R_1} \cdot (R_2 \cdot F(x_1, a_2, x_3) + \overline{R_2} \cdot F(x_1, x_2, x_3))\end{aligned}$$

In order for us to retime the reset circuitry for R_1, we must have $F(a_1, x_2, x_3) = c_1$ where c_1 is a constant. In this case a_1 is said to be a controlling value for input R_1. Since $F(a_1, a_2, x_3) = c_1$ the above expansion simplifies to:

$$\mathcal{F} = R_1 \cdot c_1 + \overline{R_1} \cdot (R_2 \cdot F(x_1, a_2, x_3) + \overline{R_2} \cdot F(x_1, x_2, x_3))$$

This permits us to move the multiplexor for R_1 to the outputs of F. If $F(a_1, x_2, x_3) \neq c$, then we cannot retime the reset circuitry for R_1. Similarly, if $F(x_1, a_2, x_3) = c_2$ (c_2 is a constant), then we can retime the reset circuitry for R_2. If both $F(a_1, x_2, x_3) = c_1$ and $F(x_1, a_2, x_3) = c_2$ are true, then $c_1 = c_2 = c$ since $F(a_1, a_2, x_3)$ must have a unique value. In this case the expression for F simplifies to:

$$\mathcal{F} = (R_1 + R_2) \cdot c + \overline{R_1} \cdot \overline{R_2} \cdot F(x_1, x_2, x_3)$$

permitting retiming of the reset circuitry for both R_1 and R_2 simultaneously.

Thus we see that in this case we cannot always guarantee that we will be able to retime the reset circuitry. It is possible only for those reset values on input lines that are controlling.

This method of retiming reset circuitry is best illustrated with some examples. Consider the five examples in Figure 11. In the figure, we are show the reset circuitry implicitly, thus assuming that the inverse of reset mapping (Φ^{-1}) has been applied after retiming. The reset circuitry for both the inputs of the NAND-gate is retimed forward. For the AND-gate, the reset circuitry for both the reset lines $R1$ and $R2$ can be retimed forward. For the OR-gate, the reset circuitry on the first input line cannot be retimed forward. For the ORAND-gate, the reset circuitry from the first two inputs, which corresponds to a common reset line $R1$, can be retimed forward; the reset circuitry for $R2$ has to be left behind. For the DEMUX, the reset circuitry for reset line $R1$ can be retimed forward; there was no reset circuitry for the second select line.

For the OR-gate example (the third example in Figure 11), it might seem that we could retime the reset circuitry forward and give it a reset value of either 0 or 1, because at circuit power-up the no-reset latch could power up in either 0 or 1. But, this retiming of the reset state is not valid because, even though at power-up the second latch can either settle to 0 or 1, we cannot assume an arbitrary value for this latch any time the reset line $R1$ is asserted later in circuit operation. This is especially important considering that the circuit may not a unique global reset line and also that often hardware reset lines are derived from other logic in the design.

Backward retiming

If different latches on the output lines have different reset lines, there is no efficient way to retime the reset circuitry backwards while preserving valid replaceability; in order to retime the latches backwards the respective reset circuitry must be left behind. Note that many gates are single-output (a notable exception is the fanout junction which we model as a single-input multi-output gate); so it may not be so important to worry about the inability of backward retiming of multi-output gates with different reset lines for different outputs. Also, it is only during forward retiming that we may hinder the space of possible retimings if we separate the reset circuitry from the latches, like in the example discussed in Section 1.

3.5 Asynchronous resets for Case C

The mapping Φ is not valid for asynchronous resets. So we cannot directly use the arguments of the previous section. We consider the cases for forward and backward retiming separately.

For forward retiming, when the input reset values are controlling, we can retime the controlling reset value forward. For example, the retimings across the NAND-gate, the AND-gate and the DEMUX-gate in Figure 11 are valid. The validity of these moves follows from an argument similar to that in Section 3.3. The only generalization is that now we have reset latches driven by different reset lines and that we have some no-reset latches as well. Since all reset values on the input latches are controlling, whenever *any* reset line is asserted, the original design will go of one of many states (depending on the current values at latches whose reset is not asserted). However, it is easy to see that all these states are equivalent to each other and to the unique state the retimed design goes to when any reset line is asserted (with an argument similar to the one used in proving Lemma 3). Unfortunately, for the case where any input reset value is non-controlling, it not possible to retime the latches forward; note that since Φ is not valid, it is not even possible to just retime no-reset latches forward and leave the reset circuitry behind. Thus, it is not possible to achieve the retimings for the OR-gate or the ORAND-gate in Figure 11. There is a way to retime the latches if we are willing to duplicate the function F across which the latches are retimed, but we omit it for brevity.

For backward retiming, although it is not possible to retime the

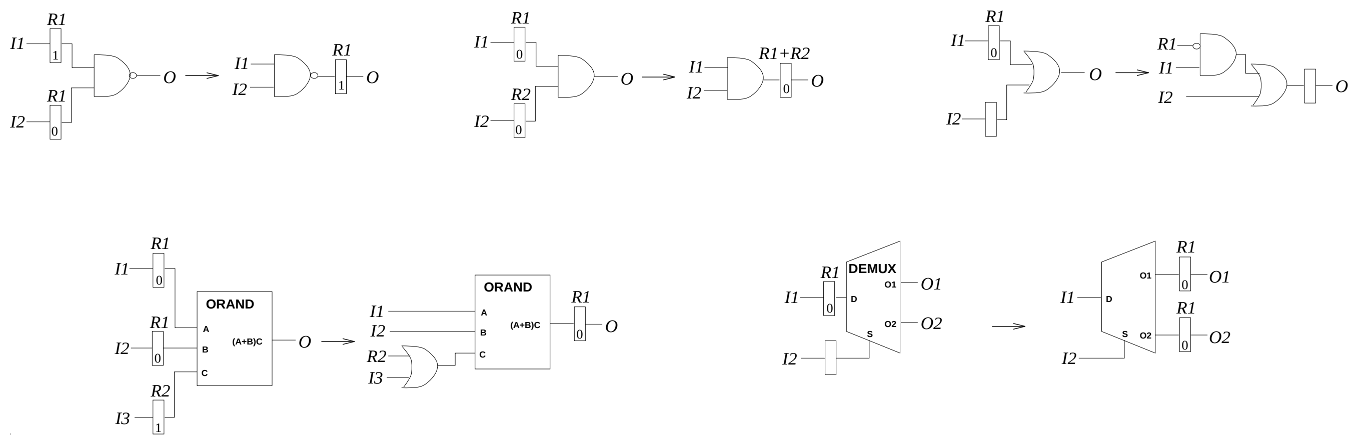

Figure 11: Examples of forward retiming of multiple reset lines.

reset values backward, we can retime the no-reset latches and leave a combinational subcircuit (corresponding to the reset circuitry for the asynchronous case) behind, just like in Figure 10 (except that we have multiple reset latches, one corresponding to each reset line). The proof is similar to that for Proposition 5.

4 Summary

We describe an overall strategy for retiming the reset state for use in a synthesis tool. We show that reset latches can be represented and retimed in the ER framework effectively. We show that for the most common reset models (single reset signal), this neither restricts the space of retiming moves nor adds unnecessary extra logic or wiring as conjectured before. Whenever possible, the no-reset latch plus its associated reset circuitry can be thought of as one unit (and a simple post-processing tool can easily undo the reset mapping to obtain reset latches from no-reset latches plus the reset circuitry). If possible, we retime this whole unit together. In case it is not possible to retime the reset state, the reset circuitry can be left behind and just the no-reset latch retimed.

To summarize, we make a strong case to model the reset circuitry explicitly at the technology-independent optimization stage (which includes the retiming operations). This does not restrict the space of retiming transformations, and the lost area can likely be recovered at the technology mapping stage. In design situations where routing the global reset line is an issue, the ER framework gives additional flexibility in trying to minimize the routing of reset circuitry by separating the reset circuitry from latches and retiming the latches. Also, for designs where the technology library contains logic elements where each implements a latch plus some combinational logic [4], the techniques described in this paper should be useful in “retiming” such elements.

5 Acknowledgments

This research was prompted by queries from Rick Rudell of Synopsys, Inc. and Leon Stok of IBM Research Center. We also thank Ellen Sentovich of Ecole des Mines de Paris for many helpful discussions.

References

[1] K. Bartlett, G. Borriello, and S. Raju. Timing Optimization of Multiphase Sequential Logic. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, 10(1):51–62, January 1991.

[2] G. De Micheli. Synchronous Logic Synthesis: Algorithms for Cycle-Time Minimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, 10(1):63–73, January 1991.

[3] G. Even, I. Y. Spillinger, and L. Stok. Retiming Revisited and Reversed. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, 15(3):348–357, March 1996.

[4] J. Grodstein, E. Lehman, H. Harkness, H. Touati, and B. Grundmann. Optimal Latch Mapping and Retiming within a Tree. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 242–245, San Jose, CA, November 1994.

[5] M. A. Iyer, D. E. Long, and M. Abramovici. Identifying Sequential Redundancies Without Search. In *Proc. of the Design Automation Conf.*, pages 457–462, Las Vegas, NV, June 1996.

[6] C. E. Leiserson and J. B. Saxe. Optimizing Synchronous Systems. *Journal of VLSI and Computer Systems*, 1(1):41–67, Spring 1983.

[7] S. Malik, E. M. Sentovich, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Retiming and Resynthesis: Optimization of Sequential Networks with Combinational Techniques. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, 10(1):74–84, January 1991.

[8] Sharad Malik. *Combinational Logic Optimization Techniques in Sequential Logic Synthesis.* PhD thesis, Electronics Research Laboratory, University of California, Berkeley, CA 94720, November 1990. Memorandum No. UCB/ERL M90/115.

[9] N. Shenoy and R. Rudell. Efficient Implementation of Retiming. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 226–233, San Jose, CA, November 1994.

[10] V. Singhal, C. Pixley, A. Aziz, and R. K. Brayton. Exploiting Power-up Delay for Sequential Optimization. In *Proc. European Design Automation Conf.*, pages 54–59, Brighton, Great Britain, September 1995.

[11] H. J. Touati and R. K. Brayton. Computing the Initial States of Retimed Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, 12(1):157–162, January 1993.