On the Optimization Power of Retiming and Resynthesis Transformations

Rajeev K. Ranjan* Vigyan Singhal† Fabio Somenzi‡ Robert K. Brayton§

Abstract

Retiming and resynthesis transformations can be used for optimizing the area, power, and delay of sequential circuits. Even though this technique has been known for more than a decade, its exact optimization capability has not been formally established. We show that retiming and resynthesis can exactly implement *1-step equivalent* state transition graph transformations. This result is the strongest to date. We also show how the notions of retiming and resynthesis can be moderately extended to achieve more powerful state transition graph transformations. Our work will provide theoretical foundation for practical retiming and resynthesis based optimization and verification.

1 Introduction

In combinational synthesis [1, 8], the positions of the latches are fixed and the logic is optimized for area, delay, or power. In retiming [5], the latches are moved across combinational gates. Retiming can change the number of latches (and hence the area) and the minimum cycle time (i.e., the clock rate). Retiming can also change the interaction between interaction between different combinational logic blocks, so retiming followed by combinational synthesis allows logic optimization not possible by combinational optimization alone. Moreover, combinational synthesis can generate new latch locations, perhaps leading to further optimization. A sequence of retiming and combinational resynthesis steps can provide a powerful way to optimize a sequential circuit [2, 7]. Iyer *et al.* [3] used a "retiming-reencoding" method that transforms a circuit with a given encoding into a circuit with arbitrary encoding and code length, and an equivalent, but not necessarily identical state transition graph.

Even though retiming and synthesis techniques have existed for over a decade, the optimization capability of a combination of these transformations has not been formally established. Given a circuit, the transformations brought in by retiming and resynthesis operations can be analyzed at the structural level. However, structural analysis gives only a local view of the transformation. A more global approach is to analyze the related STG transformations since there could be many circuit implementations of a given STG. Malik [6] characterizes the optimization capability of retiming and resynthesis by relating them to STG transformations. In particular, he states a certain subset of STG transformations (*"non-CP"*) can be implemented using retiming and resynthesis. We prove the above classification result to be incorrect by two examples. We show that the mistake in the proof reduces to the notion of equivalent states which can be merged, split, or switched in STG transformations. Our first contribution is to correct the above result and towards that we establish that iterative retiming and resynthesis can implement a different subset ("*1-step equivalent*") of STG transformations. A more significant contribution of our work is proving the converse of this result, i.e., we show that the STGs of the original circuit and of the transformed circuit are *1-step equivalent*. To our knowledge this is the first result which gives a complete and tight bound on the optimization capability of retiming and resynthesis transformations. Here it is worth mentioning that in this work we are concerning ourselves only with theoretical bounds on the optimization potential and not with actual algorithm which could achieve the optimization.

The rest of the paper is organized as follows. We present preliminary material and establish our terminology in Section 2. In Section 3 we present the exposition on the optimization power as given in [6]. Section 4 forms the core of the paper. We indicate the mistake in the exposition using counter-examples. We also correct and extend the results on the relationship between retiming and resynthesis steps and related STG transformations. In Section 5, we discuss simple extensions to traditional notions of combinational optimization and retiming which can improve their optimization capability without a significant increase in the algorithm complexity.

2 Preliminaries

In this section, we establish our circuit model and make our notions of retiming and combinational synthesis precise.

Circuit model: A sequential circuit is an interconnection of combinational gates (no combinational cycles) and memory elements along with input and output ports.

* Synopsys Inc., Mountain View, CA

† Cadence Berkeley Labs, Berkeley, CA

‡ Dept. of ECE, University of Colorado, Boulder, CO, Cadence Berkeley Labs, Berkeley, CA

§ Department of EECS, University of California, Berkeley, CA

Typically various notions of sequential circuits differ in the definition of the memory elements. We focus on sequential circuits where all the memory elements are edge-triggered latches driven by the same clock.

Combinational synthesis: In this optimization only combinational part of the circuit is changed. Any logic optimization which preserves the I/O functionality between the combinational outputs (primary outputs and latch inputs) and the combinational inputs (primary inputs and latch outputs) falls under combinational synthesis.

Retiming: Retiming is the operation of assigning lag values to each combinational gate which corresponds to the number of latches moved from the outputs to the inputs of the gate (a negative lag value indicates the latch movement from the inputs to the outputs) [5].

State transition graph(STG)**:** An STG is a labeled directed graph $G(V, \vec{E})$, where each vertex $v \in V$ corresponds to a state s, defined by the values of the latches, of the circuit. An edge $e_{ij} \in \vec{E}$ with label a connects v_i to v_j if the circuit transitions from state s_i to s_j on primary input a.

3 Malik's Results

In this section we present the results given by Malik [6][1]. The two theorems presented here consider the cases of identical and distinct STGs.

The following theorem asserts the state encoding power of retiming and resynthesis operations.

Theorem 3.1 (Encoding power of retiming and resynthesis) *Given a machine implementation M_1, corresponding to a state transition graph G, with a state assignment S_1, it is always possible to derive a machine M_2 corresponding to the same state transition graph G, and a state assignment S_2 by applying only a series of resynthesis and retiming operations on M_1.*

The proof of the theorem made use of one-to-one mapping between the states of M_1 and M_2, thereby transforming one state assignment to another using appropriate logic.

Malik also discussed the case where the STGs of M_1 and M_2 are different. It is asserted that under restricted state-transformations of the STG, the final circuit can be obtained from the initial circuit using retiming and resynthesis operations. The following basic transformations are introduced towards establishing the result. Suppose G_1 and G_2 are STGs corresponding to M_1 and M_2 respectively. G_1 may be modified to obtain G_2 through a series of three basic transformations. These transformations may create states that are equivalent to existing states, merge states that are equivalent to each other, and modify state transitions to go to states equivalent to the original destinations. The definitions of basic transformations are given below:

2-way split A state s_1 in G_1 is split to two equivalent states in G_2 (Figure 1a).

2-way merge Two equivalent states s_{11} and s_{12} in G_1 are merged to a single state s_1 in G_2 (Figure 1a).

Switch A transition in G_1 to a state s_{11} is modified to go to an equivalent state s_{12} in G_2 (Figure 1b).

The 2-way split and 2-way merge constitute *primitive transformations*, a 2-way switch, multi-way splits and merges can be accomplished by a sequence of 2-way splits and merges (Figure 1c).

Definition 1 *A* **labeled cycle of equivalent states** *in an* STG *is a directed cycle such that all state vertices in the cycle are equivalent and all transition predicate vectors on the edges in the cycle have the same label.*

Definition 2 *A* **cycle preserving** (CP) **transformation** *does not create or destroy a labeled cycle of equivalent states.*

A *non cycle preserving transformation* (NON-CP) creates or destroys a labeled cycle of equivalent states.

Theorem 3.2 *Let M_1 be an implementation corresponding to the state assignment S_1 and* STG *G_1 and M_2 be an implementation corresponding to the state assignment S_2 and* STG *G_2. If G_2 is obtained from G_1 using only* CP *transformations then M_2 can be obtained from M_1 using only a sequence of retiming and resynthesis operations.*

The proof considered G_2 to contain a CP 2-way split of some state s_1 in G_1. A transition to s_1 in G_1 corresponds to a transition to either s_{11} or s_{12} in M_2 depending on the primary input vector. It was stated that the primary input vector and state s_1 uniquely determine which of s_{11} or s_{12} is the destination state in M_2. Thus, the one-to-many mapping between the state codes for M_1 and the state codes for M_2 is actually a one-to-one mapping between the M_1 state codes plus the primary input and M_2 state codes. This can be accomplished through a combinational circuit C. Circuit C' performs many-to-one mapping from M_2's state codes to M_1's state codes. The proof was illustrated with a figure that is reproduced in Figure 2. The figure shows how the circuit may be retimed resulting in a circuit that corresponds to G_2. This may be further resynthesized to any circuit M_2 that corresponds to state assignment S_2.

4 Our Contribution

In this section we develop our result which establishes the relationship between retiming and resynthesis steps and the STG transformations. In Section 4.1, we establish the class

[1] These results partially appeared in [7] as well.

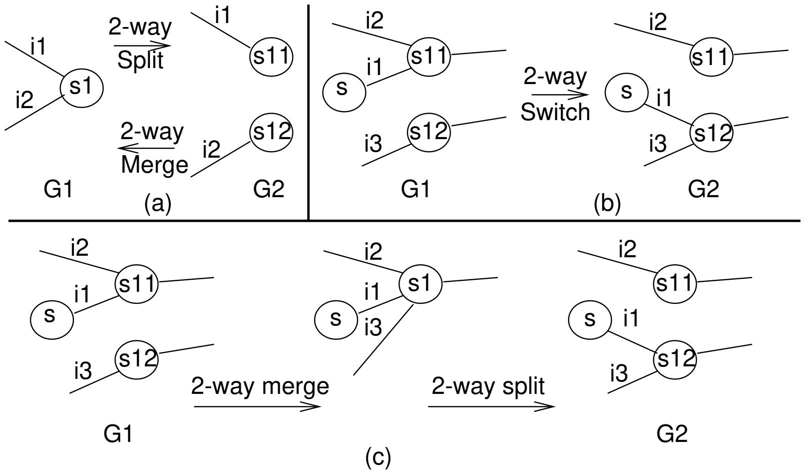

Figure 1: State-graph transformations (a) 2-way split and 2-way merge (b) switch (c) switch using 2-way split and merge.

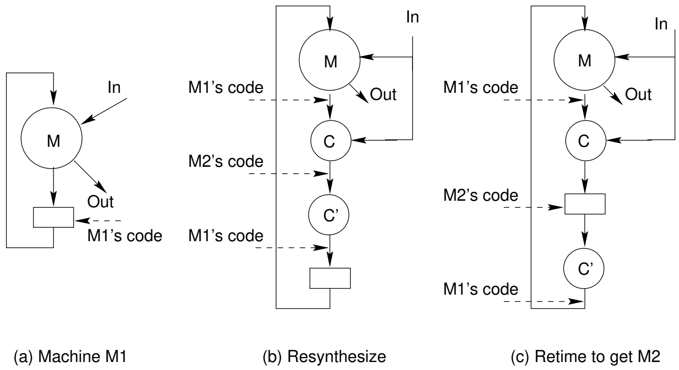

Figure 2: Obtaining equivalent FSM implementations (proof for Theorem 3.2).

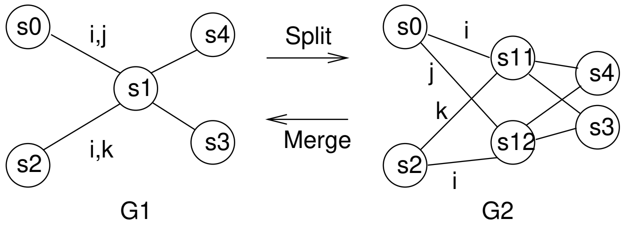

Figure 3: Counter-example to the proof of the Theorem 3.2. The next state in G_2 cannot be determined solely by the next state in G_1 and the input vector.

of STG transformations which can be implemented by retiming and resynthesis and in Section 4.2 we prove the STG transformations resulting from retiming and resynthesis operations.

4.1 STG Transformations and Retiming–Resynthesis

We begin with indicating the errors in Malik's theorem (Theorem 3.2) using some counter-examples. After identifying the sources of these errors, we provide our modifications to the proof and the theorem and establish the result.

4.1.1 Errors in Malik's Exposition

Below we given two counter-examples to the Theorem 3.2.

The proof of the Theorem 3.2 assumes that given the destination state s_1 in M_1 and primary input vector which leads to the transition, destination state in M_2 (one of s_{11} or s_{12}) can be uniquely determined. This is not correct. Consider the splitting of s_1 in G_1 as shown in the Figure 3. Given i and s_1, we cannot determine which of s_{11} or s_{12} is the next state in G_2.

The other counter-example is shown in Figure 4 [10]. The original circuit with the associated state transition graph G_1 is shown on the left. A sequentially equivalent circuit is shown on the right with the corresponding state transition graph G_2. It can be proven that neither the latches can be retimed nor can the logic be optimized indicating that a sequence of retiming and resynthesis moves cannot make this circuit transformation [10]. However, Figure 5 shows a sequence of CP transformations which transform G_1 into G_2 contradicting Theorem 3.2.

4.1.2 Implementing Splitting of a State

We modify the transformations given in the proof for Theorem 3.2, to handle the problem shown in Figure 3. The modified transformation is shown in Figure 6. The main difference between this and the transformations shown in Figure 2 is that we also make use of previous state information of M_1 in evaluating the state codes for M_2. By using information about previous state in M_1, next state in M_1, and the input, we can uniquely determine the next state for M_2. The combinational logic C' performs many-to-one mapping from M_2's state codes to M_1's state codes.

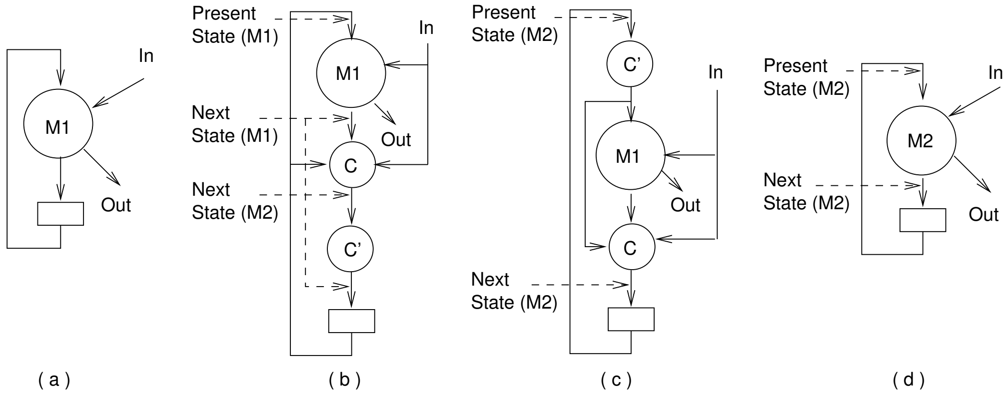

Figure 6: Illustration of STG transformation (splitting of states) which can be implemented by retiming and combinational optimization: (a) Original machine M_1 (b) Generation of next state bits for the new machine (c) Retiming to generate next state bits (d) Combinational optimization to obtain new machine M_2.

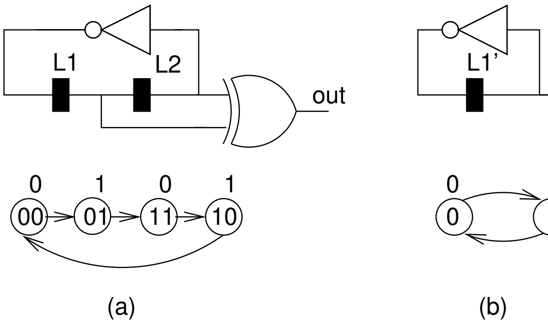

Figure 4: Counterexample to the Theorem 3.2: Original circuit in (a) cannot be transformed to the final circuit in (b) using retiming and resynthesis. The outputs are shown in boxes.

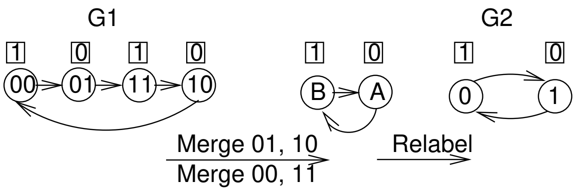

Figure 5: Using CP transformations to obtain the final STG from initial STG.

4.1.3 What STG Transformations can be Implemented by Retiming and Resynthesis?

Upon investigation we found that the problem with Theorem 3.2 lay in the notion of equivalence of states and the related permissible transformations. Below, we give the appropriate modifications and extensions.

Definition 3 *For a given* STG *two states s_1 and s_2 are* **1-step equivalent***, if they have the same output and if for all inputs i, the next state of s_1 on i is the same as the next state of s_2 on i and vice versa.*

Note that this notion of state equivalence is very local. In particular computing this equivalence does not require any fix-point computation, e.g., reachability analysis.

Based on the notion of 1-step equivalence, we modify the meanings of 2-way transformations (as given in Section 3), in the following way:

2-way split A state s_1 in G_1 is split into two 1-step equivalent states in G_2. This also includes the splitting a state with a self-loop into two 1-step equivalent states.

2-way merge Two 1-step equivalent states s_{11} and s_{12} in G_1 are merged to a single state s_1 in G_2.

Switch A transition in G_1 to a state s_{11} is modified to go to a 1-step equivalent state s_{12} in G_2.

The 2-way split and 2-way merge constitute *primitive transformations*, a 2-way switch, multi-way splits and merges can be accomplished by a sequence of 2-way splits and merges.

Definition 4 *A transformation of an* STG *G_1 into another* STG *G_2 is a* **1-step equivalent transformation** *if G_2 has been obtained from G_1 by either splitting a state into 1-step equivalent states, or merging two 1-step equivalent states, or switching between two states that are 1-step equivalent.*

Theorem 4.1 *Let M_1 be an implementation corresponding to state assignment S_1 and* STG *G_1 and M_2 be an implementation corresponding to state assignment S_2 and* STG *G_2 such that G_2 is obtained from G_1 by a* 1-STEP EQUIVALENT TRANSFORMATION. *Then M_2 can be obtained from M_1 using a sequence of retiming and resynthesis operations.*

Proof:
The proof goes along the lines of Theorem 3.2. Suppose G_2 is obtained from G_1 by splitting of some state into 1-step equivalent states. Figure 6 illustrates how splitting of 1-step equivalent states can be implemented using retiming and resynthesis. Hence M_2 can be implemented from M_1 using retiming and resynthesis operations.

Since each step in the transformation in Figure 6 is reversible, M_1 can be obtained form M_2 using retiming and resynthesis operations. This amounts to merging of 1-step equivalent states in G_2 to give G_1.

Switching between two 1-step equivalent states can be implemented by a combination of merging the two states and splitting as shown in Figure 1. ■

Definition 5 *Two* STG*s G_1 and G_2 are* 1-STEP EQUIVALENT *if one can be obtained from other by a sequence of 1-step equivalent transformations.*

Note that retiming sometimes results in transient states. In the presence of such states we use the notion of *sufficiently old configuration* [5] or *delayed designs* [9] and ignore them for the purpose of analysis.

The following theorem follows by applying induction on Theorem 4.1.

Theorem 4.2 *Let M_1 be an implementation corresponding to state assignment S_1 and* STG *G_1 and M_2 be an implementation corresponding to state assignment S_2 and* STG *G_2 such that G_2 is* 1-STEP EQUIVALENT *to G_1. Then M_2 can be obtained from M_1 using only a sequence of retiming and resynthesis operations.*

4.2 What STG transformations are Generated by Retiming and Resynthesis

In this section, we show retiming and resynthesis operations **only** result in STG transformations which are 1-STEP EQUIVALENT. Towards that direction, we make use of the following lemmas.

Lemma 4.1 *A general retiming operation as defined in Section 2, can be constructed as the sequence of retiming moves across primitive elements as shown in Figure 7.*

Proof:
The proof follows from the fact that for any circuit with complex combinational gates, there is a finite equivalent representation using NAND gates and the fanout junctions. Hence retiming across a gate is equivalent to sequence of retiming moves across the primitive elements constituting the gate. ■

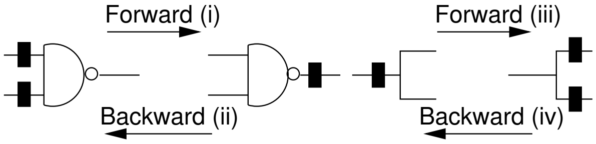

Figure 7: Primitive retiming operations. All general retiming operations can be built from a sequence of these.

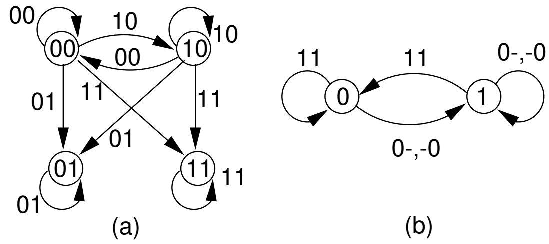

Figure 8: STG transformation when the latches are moved from the inputs (a) to the output of a NAND gate (b). Only partial STG is shown in a. The outputs are shown in boxes.

Note that this notion is similar to the atomic retiming moves considered by Singhal *et al.* [9] (NAND and fanout gate being equivalent to "justifiable" and "non-justifiable" element respectively).

Lemma 4.2 *The basic retiming operations as shown in Figure 7 result in* STG *transformations that are* 1-STEP EQUIVALENT.

Proof:
Consider the forward and backward retiming operations across the NAND gate. The corresponding STG transformations are shown in Figure 8. For clarity's sake, only a partial set of edges are shown in this figure. States $(00, 10, 01)$ are pairwise 1-STEP EQUIVALENT. The STG on the right can be obtained by merging these three states into a single one. Hence retiming across a NAND gate results in 1-STEP EQUIVALENT STG transformations.

Now consider the forward and backward retiming operations across the fanout gate. Moving latches to the output of

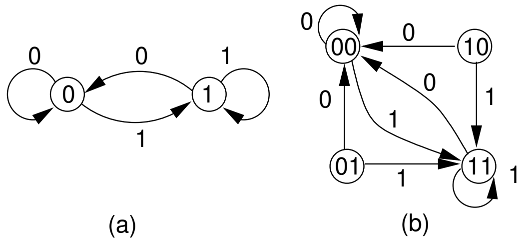

Figure 9: STG transformation when the latches are moved from the input (a) to the outputs of a fanout gate (b).

the fanout gate results in two transient states as shown in Figure 9. Ignoring these transient states, the STGs in Figure 9 are isomorphic (see the discussion for Definition 5). ■

Lemma 4.3 *Suppose* STG*s G_1 and G_2 are 1-step equivalent, then $G_1 \times G$ is 1-step equivalent to $G_2 \times G$ for all G, where $\times$ represents the composition operation.*

Proof:
Suppose G_2 is obtained by splitting of a state in G_1. Suppose s_1 in G_1 splits into two states s_{11} and s_{12}. Now for every state $\{s_1, s\}$ in $G \times G_1$, there will be two states $\{s_{11}, s\}$ and $\{s_{12}, s\}$ in $G \times G_2$. Since s_1, s_{11}, and s_{12} are 1-STEP EQUIVALENT, all of them go to the identical next state for the same input, i.e., $\forall i, (\{s_1, s\}, i) = (\{s_{11}, s\}, i) = (\{s_{12}, s\}, i)$. Hence, $\{s_1, s\}$ is 1-STEP EQUIVALENT with $\{s_{11}, s\}$ and $\{s_{12}, s\}$. The merge transformation follows since, G_1 is obtained from G_2 by merge of two 1-STEP EQUIVALENT states. ■

Lemma 4.4 *Given a circuit C consisting of* NAND *gates, fanout gates, and latches. Suppose C' is the new circuit obtained by performing one of the primitive retiming operations shown in Figure 7. If G and G' are the* STG*s of C and C' respectively, then G' is* 1-STEP EQUIVALENT *to G.*

Proof:
Suppose X is primitive gate (NAND or fanout) involved in the retiming operation. Think of the original circuit as composition of gate X with the rest of the circuit. Suppose G_{C_X} is the STG for the primitive gate and $G_{\bar{C}_X}$ is the STG for the rest of the circuit, so $G = G_{C_X} \times G_{\bar{C}_X}$. If $G_{\tilde{C}_X}$ is the STG for the primitive gate after retiming, then $G' = G_{\tilde{C}_X} \times G_{\bar{C}_X}$. Now from Lemma 4.2, G_{C_X} and $G_{\tilde{C}_X}$ are 1-STEP EQUIVALENT. Hence from Lemma 4.3, $G_{C_X} \times G_{\bar{C}_X}$ is 1-STEP EQUIVALENT to $G_{\tilde{C}_X} \times G_{\bar{C}_X}$, implying that G and G' are 1-STEP EQUIVALENT. ■

Theorem 4.3 *Suppose G_1 and G_2 are* STG*s associated with circuits M_1 and M_2 respectively such that M_2 is obtained from M_1 using a sequence of retiming and resynthesis operations. Then G_1 are G_2 are* 1-STEP EQUIVALENT.

Proof:
Combinational synthesis does not modify the STG. From Lemma 4.1, retiming corresponds to a sequence of retiming steps across primitive elements. An iterative general retiming results in concatenation of sequences consisting of retiming steps across primitive elements. From Lemma 4.4, at each step the resulting transformation is 1-STEP EQUIVALENT. Hence G_1 and G_2 are 1-STEP EQUIVALENT. ■

Theorem 4.4 *Let M_1 be an implementation corresponding to state assignment S_1 and* STG *G_1 and M_2 be an implementation corresponding to state assignment S_2 and* STG *G_2. M_2 can be obtained from M_1 using only a sequence of retiming and resynthesis operations if and only if G_1 are G_2 are* 1-STEP EQUIVALENT.

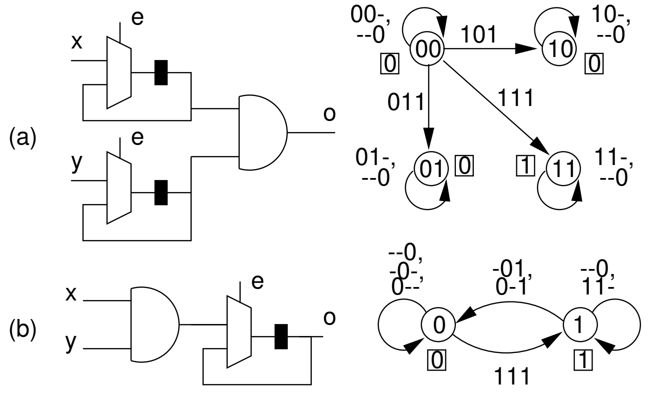

Figure 10: Original circuit in (a) cannot be transformed to the final circuit in (b) using retiming and resynthesis. For clarity purposes only partial set of edges is shown for circuit a. The outputs are shown in boxes. The order of input labels on edges is (x, y, e).

Proof: Directly follows from Theorems 4.2 and 4.3. ■

In view of this theorem, we make following observations.

- The counter-example described in Section 4.1.1 can be explained in the following way. The transformation in Figure 5 involves merging the state "01" with "10" and state "00" with "11". However, since these states are not 1-step equivalent, the STG transformation cannot be implemented with retiming and resynthesis transformations.

- Since 1-STEP EQUIVALENCE is a local notion, intuitively 1-STEP EQUIVALENT TRANSFORMATIONS cover only a small subset of all valid STG transformations. For example, Figure 10 shows two circuits along with their STGs (only partial STG is shown for the circuit a). The circuits in Figures 10a and b are sequentially equivalent, but one cannot be obtained from the other using retiming and synthesis transformations. This is because all three equivalent states $(00, 10, 01)$ have self-loops with predicate $(- - 0)$, implying they are not 1-STEP EQUIVALENT. Hence their merger cannot be implemented using only retiming and resynthesis transformations.

- We do not need the condition of CP preserving transformations. The counterexample is shown in Figure 11. In STG G_1, the self-loop on s_1 is a *labeled cycle of equivalent states* (there is just one state in the cycle). However, in STG G_2, due to the self-loops on s_{11} and s_{12}, we have two labeled cycles of equivalent states, i.e., this STG transformation is non-CP. However, as shown in Figure 6 we can implement splitting of states using retiming and resynthesis.

  Next consider Malik's examples of non-CP [6] shown in Figure 12. The merger of states s_{11} and s_{12} in Figure 12a is not a valid 2-way merge because states s_{11}

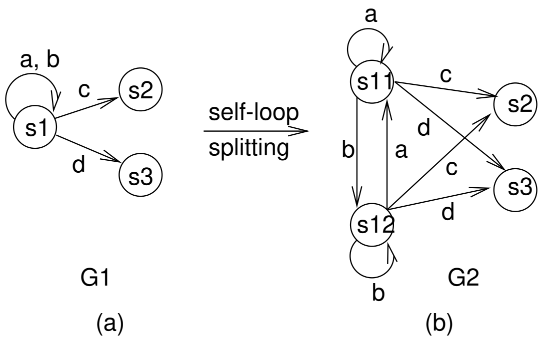

Figure 11: STG transformations involving splitting a state with a self-loop.

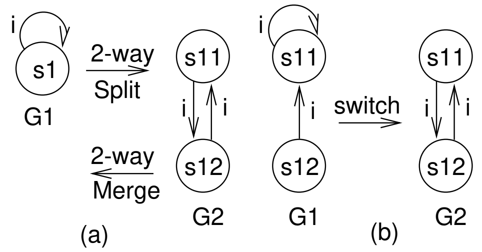

Figure 12: Non-CP transformations.

and s_{12} are not 1-step equivalent. In Figure 12b, the transformation involves a switch. Notice that states s_{11} and s_{12} are 1-step equivalent. However, after the switch, states s_{11} and s_{12} are no longer 1-step equivalent, making the switch transformation invalid.

5 Extending Notions of Retiming and Synthesis

The examples given in the previous section illustrate the limitations of retiming and combinational transformations. In this section we show how to increase the optimization capability of these transformations by extending the notions of conventional retiming and combinational optimization.

5.1 Eliminating Floating Latches

The current combinational optimization techniques do little manipulation of latches (e.g., latch removal via constant propagation). While gates that do not transitively fanout to any primary output are eliminated during combinational optimization, latches are treated as pseudo primary inputs and outputs and hence are not eliminated even if they do not transitively fanout to any primary output. Such latches also are not eliminated during a retiming operation either. We can extend the notion of combinational optimization to one which trivially gets rid of such latches before proceeding to regular combinational optimization. The process of removing latches that do not fanout to any primary output is termed as

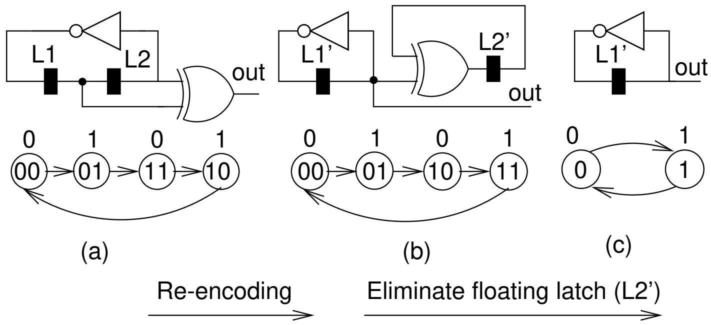

Figure 13: Circuit transformation using floating latch elimination.

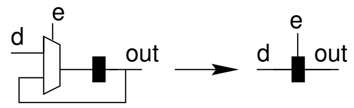

Figure 14: A latch with a feedback path can be modeled as an enabled latch.

floating latch elimination. It does not add to the complexity of the synthesis algorithm. With this extended notion of synthesis, the circuit transformation shown in Figure 4 can be obtained. The transformation process is shown in Figure 13. Essentially, the first transformation re-encodes the circuit, which can be implemented by retiming and resynthesis as explained in Theorem 3.1. This is followed by floating latch elimination.

In general, this transformation will allow us to implement STG transformations, where a circuit is properly reencoded to expose redundant state bits that can be eliminated.

5.2 Retiming Latches with Latch Enable Signal

The direct feedback path to the latch in the circuit of Figure 10 can be thought of as an enabled latch as shown in the Figure 14. In [4], a retiming technique is proposed to handle circuits containing edge-triggered latches with different enable signals and different clocks. The retiming problem for multiple-class sequential circuits was reduced to an equivalent retiming for single class sequential circuits, thereby exploiting performance enhancements made in that domain. In particular, their technique would allow the retiming move as shown in Figure 15. With this extension to the retiming move, we can obtain the transformation shown in Figure 10.

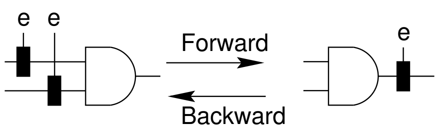

Figure 15: Retiming enabled-latch across gates.

This extension to the notion of retiming enables us to merge states which are 1-STEP EQUIVALENT except for the identical predicate on the self-loop, i.e., the state holds its value for a particular input combination.

6 Conclusion

Retiming and resynthesis are powerful tools to optimize a sequential circuit. In this work, we have formally characterized the optimization capability of retiming and resynthesis steps in terms of the transformations on the respective STGs of the circuits. We have shown that retiming and resynthesis steps are exactly the *1-step equivalent* transformations on STGs. To our knowledge this is the first result which gives a complete and tight bound on the optimization capability of retiming and resynthesis transformations.

We have demonstrated that by simple extensions to traditional notions of retiming and combinational optimization, we can achieve more complex STG transformations. These extensions do not result in increased algorithmic complexity of optimization steps. It will be an interesting exercise to obtain establish similar tight bounds on the optimization capability of these extended notions.

References

[1] R. K. Brayton, R. Rudell, A. L. Sangiovanni-Vincentelli, and A. R. Wang. MIS: A Multiple-Level Logic Optimization System. *IEEE Trans. Comput.-Aided Design Integrated Circuits*, CAD-6(6):1062–81, Nov. 1987.

[2] S. Hassoun and C. Ebling. Sequential Circuit Optimization Using Precomputation. In *Proc. IEEE/ACM Intl. Workshop on Logic Synthesis*, May 1997.

[3] B. Iyer and M. Ciesielski. Metamorphosis: State Assignment by Retiming and Re-encoding. In *Proc. IEEE/ACM International Conference on Computer-Aided Design*, pages 614–7, 1996.

[4] C. Legl, P. Vanbekbergen, and A. Wang. Retiming of Edge-Triggered Circuits with Mulitple Clocks and Load Enables. In *Proc. IEEE/ACM Intl. Workshop on Logic Synthesis*, 1997.

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

[6] S. Malik. *Combinational Logic Optimization Techniques in Sequential Logic Synthesis*. PhD thesis, University of California Berkeley, Nov. 1990. Memorandum No. UCB/ERL M90/115.

[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 Trans. Comput.-Aided Design Integrated Circuits*, 10(1):74–84, Jan. 1991.

[8] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. SIS: A System for Sequential Circuit Synthesis. Technical Report UCB/ERL M92/41, Electronics Research Lab, Univ. of California, Berkeley, CA 94720, May 1992.

[9] V. Singhal, C. Pixley, R. L. Rudell, and R. K. Brayton. The Validity of Retiming Sequential Circuits. In *Proc. of the IEEE/ACM Design Automation Conf.*, pages 316–21, June 1995.

[10] H. Zhou, V. Singhal, and A. Aziz. How Powerful is Retiming? In *Proc. IEEE/ACM Intl. Workshop on Logic Synthesis*, May 1998. to appear.