Retiming and Resynthesis: A Complexity Perspective

Jie-Hong R. Jiang, *Member, IEEE*, and Robert K. Brayton, *Fellow, IEEE*

***Abstract*— Transformations using retiming and resynthesis operations are the most important and practical (if not the only) techniques used in optimizing synchronous hardware systems. Although these transformations have been studied extensively for over a decade, questions about their *optimization capability* and *verification complexity* are not answered fully. Resolving these questions may be crucial in developing more effective synthesis and verification algorithms. This paper settles the above two open problems. The optimization potential is resolved through a constructive algorithm which determines if two given finite state machines (FSMs) are transformable to each other via retiming and resynthesis operations. Verifying the equivalence of two FSMs under such transformations, when the history of iterative transformation is unknown, is proved to be PSPACE-complete and hence just as hard as general equivalence checking, contrary to a common belief. As a result, we advocate a conservative design methodology for the optimization of synchronous hardware systems to ameliorate verifiability. Our analysis reveals some properties about initializing FSMs transformed under retiming and resynthesis. On the positive side, a lag-independent bound is established on the length increase of initialization sequences for FSMs under retiming. It allows a simpler incremental construction of initialization sequences compared to prior approaches. On the negative side, we show that there is no analogous transformation-independent bound when resynthesis and retiming are iterated. Nonetheless, an algorithm computing the exact length increase is presented.**

***Index Terms*— Computational Complexity, Equivalence Verification, Finite State Machine, Initialization Sequence, Retiming, Resynthesis.**

I. Introduction

RETIMING [9], [10] is an elementary yet effective technique in optimizing synchronous hardware systems. By simply repositioning registers, it is capable of rescheduling computation tasks in an optimal way subject to some design criteria. As both an advantage and a disadvantage, retiming preserves the circuit structure of the system under consideration. It is an advantage in that it supports incremental engineering change with good predictability, and a disadvantage in that the optimization capability is somewhat limited. Therefore, resynthesis [13], [1], [14] was proposed to be combined with retiming, allowing modification of circuit structures. This combination of retiming and resynthesis certainly extends the optimization power of retiming, but to what extent remains an open problem, even though some notable progress has been made since [13], e.g. [18], [19], [25]. Fully resolving this problem is crucial in understanding the complexity of verifying the equivalence of systems transformed by retiming and resynthesis and in constructing correct initialization sequences. In fact, despite its effectiveness, the transformation of retiming and resynthesis is not widely used in hardware synthesis flows due to the verification hindrance and the initialization problem. Progress in these areas could enhance the practicality and application of retiming and resynthesis, and advance the development of more effective synthesis and verification algorithms.

This paper tackles three main problems regarding retiming and resynthesis:

Optimization power:
: What is the transformation power of retiming and resynthesis? How can we tell if two synchronous systems are transformable to each other with retiming and resynthesis operations?

Verification complexity:
: What is the computational complexity of verifying if two synchronous systems are equivalent under retiming and resynthesis?

Initialization:
: How does the transformation of retiming and resynthesis affect the initialization of a synchronous system? How can we correct initialization sequences?

Our main results include

- (Section III) Characterize constructively the transformation power of retiming and resynthesis.
- (Section IV) Prove the PSPACE-completeness of verifying the equivalence of systems transformed by retiming and resynthesis operations when the transformation history is lost.
- (Section V) Demonstrate the effects of retiming and resynthesis on the initialization sequences of synchronous systems. Present an algorithm correcting initialization sequences.

The paper is organized as follows. After Section II introduces some preliminaries and notation, our main results are presented in Sections III, IV, and V. In Section VI, a closer comparison with prior work is detailed. Section VII concludes this paper and outlines some future research directions.

J.-H. Jiang is with the Department of Electrical Engineering and the Graduate Institute of Electronics Engineering, National Taiwan University, Taipei 10617, Taiwan.

R. Brayton is with the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720, USA.

A preliminary version of this paper appeared in [5]. This work was supported in part by NSC grant 94-2218-E-002-083, NSF grant CCR-0312676, the California Micro program, and industrial sponsors, Fujitsu, Intel, Magma and Synplicity.

II. PRELIMINARIES

In this paper, to avoid later complication[1] we shall not restrict ourselves to binary variables and Boolean functions. Thus, we assume that variables can take values from arbitrary finite domains, and similarly functions can have arbitrary finite domains and co-domains. When (co)domains are immaterial in the discussion, we shall omit specifying them. We introduce the following notational conventions. Let $\mathcal{V}_1$ be a set of variables. Notation $[\![\mathcal{V}_1]\!]$ represents the set of all possible valuations over $\mathcal{V}_1$. Let $\mathcal{V}_2 \subseteq \mathcal{V}_1$. For $x \in [\![\mathcal{V}_1]\!]$, we use $x[\mathcal{V}_2] \in [\![\mathcal{V}_2]\!]$ to denote the valuation over variables $\mathcal{V}_2$ which agrees with x on $\mathcal{V}_2$. For instance, let $\mathcal{V}_1 = \{v_1, v_2, v_3\}$ and $\mathcal{V}_2 = \{v_2, v_3\}$ be two sets of Boolean variables. For valuations $x = (v_1 = 0, v_2 = 1, v_3 = 0)$ and $y = (v_1 = 1, v_2 = 1, v_3 = 0)$ over $\mathcal{V}_1$, we have $x[\mathcal{V}_2] = y[\mathcal{V}_2] = (v_2 = 1, v_3 = 0)$.

Synchronous hardware systems. Based on [9], a syntactical definition of *synchronous hardware systems* can be formulated as follows. A hardware system is abstracted as a directed graph, called a *communication graph*, $G = (V, E)$ with *typed* vertices V and *weighted* edges E. Every vertex $v \in V$ represents either the environment or a functional element. The vertex representing the environment is the *host*, which is of type undefined; a vertex is of type $\vec{f}$ if the functional element it represents is of function $\vec{f}$ (which can be a multiple-output function consisting of $f_1, f_2, \ldots$). Every edge $e\langle w\rangle = (u, v)\langle w\rangle \in E$ with a nonnegative integer-valued weight w corresponds to the interconnection from vertex u to vertex v interleaved by w state-holding elements (or registers). (From the viewpoint of hardware systems, any component in a communication graph disconnected from the host is redundant. Hence, in the sequel, we assume that a communication graph is a single connected component.) A hardware system is *synchronous* if, in its corresponding communication graph, every cycle contains at least one positive-weighted edge. This paper is concerned with synchronous hardware systems whose registers are all triggered by the same clock ticks. Moreover, according to the initialization mechanism, a register can be reset either explicitly or implicitly. For registers with explicit reset, their initial values are determined by some reset circuitry when the system is powered up. In contrast, for registers with implicit reset, their initial values can be arbitrary, but can be brought to an identified set of states (i.e. the set of *initial states*[2]) by applying some input sequences, the so-called *initialization* (or *reset*) *sequences* [17]. It turns out that explicit-reset registers can be replaced with implicit-reset ones plus some reset circuitry [14], [21]. (Doing so admits a more systematic treatment of retiming synchronous hardware systems because retiming explicit-reset registers needs special attention to maintain equivalent initial states.) Without loss of generality, this paper assumes that all registers have implicit reset. In addition, we are concerned with initializable systems, that is, there exist input sequences which bring the systems from any state to some set of designated initial states.

The semantical interpretation of synchronous hardware systems can be modelled as *finite state machines* (FSMs). An FSM $\mathcal{M}$ is a tuple $(Q, I, \Sigma, \Omega, \vec{\delta}, \vec{\lambda})$, where Q is a finite set of states, $I \subseteq Q$ is the set of initial states, Σ and Ω are the input and output alphabets, respectively, and $\vec{\delta} : \Sigma \times Q \to Q$ (respectively $\vec{\lambda} : \Sigma \times Q \to \Omega$) is the transition function (respectively output function). Let $\mathcal{V}_S$, $\mathcal{V}_I$, and $\mathcal{V}_O$ be the sets of variables that encode the states, input alphabet, and output alphabet respectively. Then $Q = [\![\mathcal{V}_S]\!]$, $\Sigma = [\![\mathcal{V}_I]\!]$ and $\Omega = [\![\mathcal{V}_O]\!]$. As a convention, for a (current-)state variable s, its primed version s' denotes the corresponding next-state variable.

To construct an FSM from a communication graph $G = (V, E)$, for the sake of convenience we build another communication graph $G' = (V', E')$ from G as follows. Initially let $V' = V$ and $E' = \{e\langle w\rangle \in E \mid w = 0, 1\}$. For each $(u_1, u_2)\langle w\rangle \in E$ with $w \geq 2$, we introduce $w - 1$ new vertices of type identity mapping to V', say $\{v_1, \ldots, v_{w-1}\}$, and add w edges $\{(u_1, v_1)\langle 1\rangle, (v_1, v_2)\langle 1\rangle, \ldots, (v_{w-2}, v_{w-1})\langle 1\rangle, (v_{w-1}, u_2)\langle 1\rangle\}$ to E'. With the so-constructed G', we can associate a current-state variable and a next-state variable to each $(u, v)\langle 1\rangle \in E'$ to denote the output and input of the register on this edge, respectively. Let the *transitive fanin cone* rooted at a non-host vertex $v \in V'$, denoted as $\mathit{TFI}(v)$, be the set of non-host vertices $u \in V'$ such that either $u = v$ or there exists $\{(u, v_1)\langle 0\rangle, (v_1, v_2)\langle 0\rangle, \ldots, (v_{i-1}, v_i)\langle 0\rangle, (v_i, v)\langle 0\rangle\} \subseteq E'$. The transition function of a state variable s is the overall function defined by the vertices in $\mathit{TFI}(t)$ for t the fanin vertex of the register associated with state variable s. Similarly, an output function is the overall function defined by the vertices in $\mathit{TFI}(t)$ for t the fanin vertex of the corresponding output variable. Since any circuit implementing an FSM can be abstracted as a communication graph, a communication graph can be seen as a realization of an FSM.

The behavior of an FSM can be described in another graphical representation, the so-called state diagram [8] or state transition graph (STG). The STG $\Gamma = (N, A)$ of an FSM $(Q, I, \Sigma, \Omega, \vec{\delta}, \vec{\lambda})$ has nodes N representing states Q and labelled arcs A representing transitions specified by $\vec{\delta}$ and $\vec{\lambda}$. A detailed construction can be found, e.g., in [8].

We define a strong form of state equivalence which will govern the study of the transformation power of retiming.

Definition 1: Given an FSM $\mathcal{M} = (Q, I, \Sigma, \Omega, \vec{\delta}, \vec{\lambda})$, two states $q_1, q_2 \in Q$ are **immediately equivalent**[3], denoted as $q_1 \cong q_2$, if $\vec{\delta}(\sigma, q_1) = \vec{\delta}(\sigma, q_2)$ and $\vec{\lambda}(\sigma, q_1) = \vec{\lambda}(\sigma, q_2)$ for any $\sigma \in \Sigma$.

Notice that $\cong$ is reflexive, symmetric, and transitive, and thus is an equivalence relation. Also, note that the immediate equivalence differs from the standard state equivalence [8], which says that two states of an FSM are equivalent if starting from either of the two states the FSM is indistinguishable in its input-output behavior.

[1] This complication comes from encoding states in binary codes. When the code size is strictly larger than the state size, the encoding may introduce an additional unreachable state space. It complicates our discussion especially when the initialization of synchronous hardware systems is concerned.

[2] When referring to "initial states," we shall mean the starting states of a system *after* initialization.

[3] The definition of immediate equivalence corresponds to the "1-step equivalence" definition of [18]. Since the latter is confusing with k-step distinguishability [6] of states in equivalence checking, we rename it differently.

Also, we define *dangling states* inductively as follows.

Definition 2: Given an FSM, a state is **dangling** if either it has no predecessor state or all of its predecessor states are dangling. All other states are **non-dangling**.

Figure 1 shows an example, where states $\{q_0, q_1\}$ are dangling and all others are non-dangling. By the above inductive definition, we know that states in a strongly connected component[4] of an STG specified by an FSM are non-dangling. However, non-dangling states need not to be in strongly connected components as illustrated in Figure 1.

The introduced three representations, communication graphs, FSMs, and STGs, are used throughout this paper to represent synchronous hardware systems. Although these representations are interchangeable, their succinctness in representing sequential systems may differ and affect the measures in complexity analysis. To represent synchronous hardware systems with FSMs, the input size is measured mainly by the length of the formulae of transition and output functions. For the communication graph representation, the input size is measured by the length of representing typed vertices and weighted edges. Since the translation between an FSM and a communication graph is often linear, these two representations of synchronous hardware systems are of similar succinctness. On the other hand, STGs are graphs whose sizes are measured by the number of vertices (states) and edges (transitions). Translating an FSM or a communication graph into an STG suffers the so-called state explosion problem since the number of states is exponential in the number of state variables. Therefore, STGs are not efficient in representing synchronous hardware systems. However, they provide a friendly data structure to conceptualize the transformation power of retiming and resynthesis. In the sequel, complexity analysis may be conducted over different representations. It is important to notice the exponential gap between the STG representation and the other two representations.

Retiming. A *retiming operation* over a synchronous hardware system consists of a series of atomic moves of registers across functional elements in either a forward or backward direction. (The relocation of registers is crucial in exploring optimal synchronous hardware systems with respect to various design criteria, such as area, performance, power, etc. As is not our focus, the exposition of retiming in the optimization perspective is omitted in this paper. Interested readers are referred to [10].) Formally speaking, retiming can be described with a *retime function* [9] over a communication graph as follows.

Definition 3: Given a communication graph $G = (V, E)$, a **retime function** $\rho : V \to \mathbb{Z}$ maps each vertex to an integer, called the **lag** of the vertex, such that $w + \rho(v) - \rho(u) \geq 0$ for any edge $(u, v)\langle w \rangle \in E$. If $\rho(\text{host}) = 0$, ρ is called **normalized**; otherwise, ρ is **unnormalized**.

Given a communication graph $G = (V, E)$, any retime function ρ over G uniquely determines a "legally" retimed communication graph $G^\dagger = (V, E^\dagger)$ in which $(u, v)\langle w \rangle \in E$ if, and only if, $(u, v)\langle w + \rho(v) - \rho(u) \rangle \in E^\dagger$. By symmetry, the retime function $-\rho$ reverses the retiming from $G^\dagger$ to G. Figure 2 shows the retime functions of a vertex v in some communication graph corresponding to atomic backward and forward moves of registers.

Retime functions can be naturally classified by calibrating their equivalences as follows.

Definition 4: Given a communication graph G, two retime functions ρ_1 and ρ_2 are equivalent if they result in the same retimed communication graph.

Proposition 1: Given a retime function ρ_1 with respect to a communication graph, let $\rho_2 = \rho_1 - c$ for some constant $c \in \mathbb{Z}$. Then ρ_1 and ρ_2 are equivalent.

Hence any retime function can be normalized. This equivalence relation, which will be useful in the study of the increase of initialization sequences due to retiming, induces a partition over retime functions. Equivalent retime functions (with respect to some communication graph) form an equivalence class.

Proposition 2: Given a communication graph G, any equivalence class of retime functions is of infinite size; any equivalence class of normalized retime functions is of size either one or infinity (only when G contains components disconnected from the host). Furthermore, any equivalence class of retime functions has a normalized member.

Resynthesis. A *resynthesis operation* over a function f rewrites the syntactical formula representation of f while maintaining its semantical functionality. Clearly, the set of all possible rewrites is infinite (but countable, namely, with the same cardinality as the set $\mathbb{N}$ of natural numbers). When a resynthesis operation is performed upon a synchronous hardware system, we shall mean that the transition and output functions of the corresponding FSM are modified in representations but preserved in functionalities. This modification in representations will be reflected in the communication graph of the system. (Again, such rewrites are usually subject to some optimization criteria. Since this is not our focus, the optimization aspects of resynthesis operations are omitted. See, e.g., [1] for further treatment.)

The effects of retiming and resynthesis on a communication graph $G = (V, E)$ are important for our later development and worth emphasis. Retiming only alters the weights (i.e. numbers of registers on edges) of edges E, whereas the vertices and their connections of G are not affected by retiming. Resynthesis, on the other hand, can change both the vertices and their connections. However, since it needs to preserve the functionalities of transition and output functions, it can only modify a purely combinational block (i.e., a set of vertices along with the zero-weight edges connecting them). Therefore, edges $E^+ \subseteq E$ with positive weights remain intact throughout resynthesis while vertices V and edges $E \setminus E^+$ can be completely changed. The optimization capabilities of retiming and resynthesis are complementary.

III. Optimization Capability

The transformation power of retiming and resynthesis can be understood best with state transition graphs (STGs) defined by

[4] A component (or an induced subgraph) of a graph is strongly connected if any (ordered or unordered) pair of vertices are connected with some path.

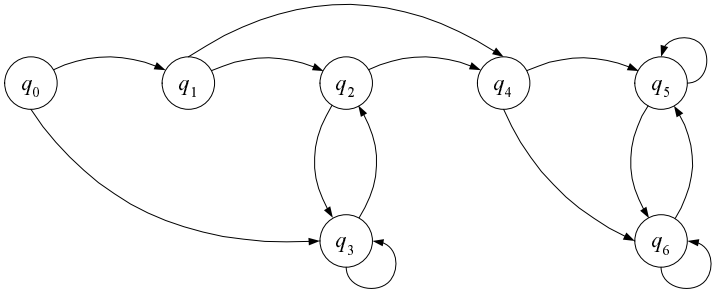

Fig. 1. In the above state transition graph (where labels on edges are omitted), states $\{q_0, q_1\}$ are dangling, and all others are non-dangling. On the other hand, states $\{q_2, q_3, q_5, q_6\}$ are in strongly connected components. Therefore, state q_4 is non-dangling but not in a strongly connected component.

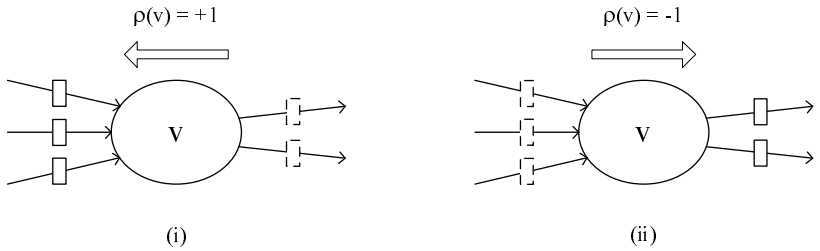

Fig. 2. Let v be a vertex in a communication graph, and boxes on edges be registers. In addition, registers to be added (respectively deleted) due to retiming are in solid (respectively dotted) boxes. (i) An atomic backward move of registers from the output edges of v to the input edges. The corresponding lag of v is $+1$. (ii) An atomic forward move of registers from the input edges of v to the output edges. The corresponding lag of v is -1.

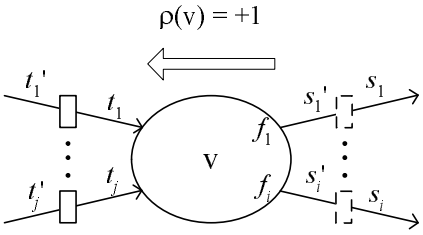

Fig. 3. Let v be a vertex of type $(f_1, \ldots, f_i)$ in some communication graph. The registers on the i output edges of v are to be retimed backward to the j input edges. Before retiming, variables $\{s_1, \ldots, s_i\}$ and $\{s'_1, \ldots, s'_i\}$ are the original current-state and next-state variables, respectively. After retiming, the input variables $\{t_1, \ldots, t_j\}$ of v become the new current-state variables.

FSMs. We investigate how retiming and resynthesis operations can alter STGs.

A. Optimization power of retiming

Given a communication graph $G = (V, E)$, we study how the atomic forward and backward moves of retiming affect the corresponding FSM $\mathcal{M} = (\llbracket \mathcal{V}_S \rrbracket, I, \Sigma, \Omega, \vec{\delta}, \vec{\lambda})$.

To study the effect of an atomic backward move, consider a normalized retime function ρ with $\rho(v) = 1$ for some vertex $v \in V$ as shown in Figure 3, and $\rho(u) = 0$ for all $u \in V \backslash \{v\}$. (Because a retiming operation can be decomposed as a series of atomic moves, analyzing ρ defined above suffices to demonstrate the effect.) Let $\mathcal{V}_S = \{s_1, \ldots, s_n\}$ be the state variables of $\mathcal{M}$. Then, according to the atomic backward move of retiming, $\mathcal{V}_S$ can be partitioned into two disjoint subsets: $\mathcal{V}_{S^\natural} = \{s_1, \ldots, s_i\}$, those changed by retiming, and $\mathcal{V}_{S^*} = \{s_{i+1}, \ldots, s_n\}$, those unchanged. Thus, $\mathcal{V}_S = \mathcal{V}_{S^\natural} \cup \mathcal{V}_{S^*}$. Moreover, suppose v is of type $\vec{f} : \llbracket \{t_1, \ldots, t_j\} \rrbracket \rightarrow \llbracket \{s'_1, \ldots, s'_i\} \rrbracket$, where the valuation of next-state variables s'_k is defined by $f_k(t_1, \ldots, t_j)$ for $k = 1, \ldots, i$. Assume that $\mathcal{M}^\dagger = (\llbracket \mathcal{V}_S^\dagger \rrbracket, I^\dagger, \Sigma, \Omega, \vec{\delta}^\dagger, \vec{\lambda}^\dagger)$ is the FSM after retiming, where state variables $\mathcal{V}_S^\dagger = \mathcal{V}_T \cup \mathcal{V}_{S^*}$ with $\mathcal{V}_T = \{t_1, \ldots, t_j\}$. For any two states $q_1^\dagger, q_2^\dagger \in \llbracket \mathcal{V}_S^\dagger \rrbracket$, if $q_1^\dagger[\mathcal{V}_{S^*}] = q_2^\dagger[\mathcal{V}_{S^*}]$ and $\vec{f}(q_1^\dagger[\mathcal{V}_T]) = \vec{f}(q_2^\dagger[\mathcal{V}_T])$, then $q_1^\dagger$ and $q_2^\dagger$ are immediately equivalent. Because $q_1^\dagger$ and $q_2^\dagger$ are mapped by $\vec{f}$ to the same value on which the transition and output functions of $\mathcal{M}^\dagger$ depend, they must have the same next state and the same output.

Comparing state pairs between $\mathcal{M}$ and $\mathcal{M}^\dagger$, there always exists a relation $R \subseteq \llbracket \mathcal{V}_S \rrbracket \times \llbracket \mathcal{V}_S^\dagger \rrbracket$ such that a state pair $(q, q^\dagger)$ is in R if, and only if, $q[\mathcal{V}_{S^*}] = q^\dagger[\mathcal{V}_{S^*}]$ and $q[\mathcal{V}_{S^\natural}] = \vec{f}(q^\dagger[\mathcal{V}_T])$. Relation R is closed under state transition, that is, if $(q, q^\dagger) \in R$, then $\vec{\lambda}(\sigma, q) = \vec{\lambda}^\dagger(\sigma, q^\dagger)$ and $(\vec{\delta}(\sigma, q), \vec{\delta}^\dagger(\sigma, q^\dagger)) \in R$ for any $\sigma \in \Sigma$. Moreover, since $\vec{f}$ is a total function, every state of $\mathcal{M}^\dagger$ has a corresponding state in $\mathcal{M}$ related by R. (It corresponds to the fact that backward moves of retiming cannot increase the length of initialization sequences, the subject to be discussed in Section V.) On the other hand, since $\vec{f}$ may not be a surjective (or an onto) mapping in general, there may be some state q of $\mathcal{M}$ such that $\forall x \in \llbracket \mathcal{V}_T \rrbracket$. $q[\mathcal{V}_{S^\natural}] \neq \vec{f}(x)$, that is, no states can transition to q. In this case, q can be seen as being annihilated after retiming. To summarize,

Lemma 1: An atomic backward move of retiming can 1) split a state into multiple immediately equivalent states and/or 2) annihilate states which have no predecessor states.

With a similar reasoning by reversing the roles of $\mathcal{M}$ and $\mathcal{M}^\dagger$, one can show

Lemma 2: An atomic forward move of retiming can 1) merge multiple immediately equivalent states into a single state and/or 2) create states which have no predecessor states. (Similar results of Lemmas 1 and 2 appeared in [19], where the phenomena of state creation and annihilation were omitted.)

Note that two immediately equivalent states, say q_1 and q_2, of an FSM may become not immediately equivalent when their common successor state splits into multiple states due to backward retiming. In this case, q_1 and q_2 may transition to different successors and become not immediately equivalent. In contrast, two non-immediately equivalent states of an FSM may possibly become immediately equivalent when their successor states are merged due to forward retiming. Therefore, retiming may not preserve the state relation of immediate equivalence. That is, this equivalence relation is not an invariant under retiming. (However, the relation of standard state equivalence [11] is an invariant even under retiming and resynthesis to be discussed in Section III-B.)

Also notice that, in a single atomic forward move of retiming, transitions among the newly created states are prohibited. In contrast, when a sequence of atomic forward moves $m_1, \ldots, m_n$ are performed, the newly created states at move m_i can possibly have predecessor states created in later moves $m_{i+1}, \ldots, m_n$. Therefore, all the newly created states not merged with original existing states by immediate equivalence are dangling. However, to be shown in Section V-A, the transition paths among these dangling states cannot be arbitrarily long.

Since a retiming operation consists of a series of commutative[5] atomic moves, Lemmas 1 and 2 set the fundamental rules of all possible changes of STGs by retiming. Observe that a retiming operation is always associated with some structure (i.e. a communication graph). For a fixed structure, a retiming operation has limited optimization power because the configurations of register positions are finite and confined to the structure. That is, there may not exist a series of atomic moves of retiming (over a communication graph) which meet arbitrary targeting changes on an STG with respect to the manipulations on immediately equivalent states. In fact, the converses of Lemmas 1 and 2 are not true (that is, there may not exist atomic moves of retiming achieving some designated state splitting, merging, creation, and/or annihilation) since one can design a communication graph in a way that the register positions are fixed and thus immediately equivalent states cannot be manipulated as desired. Figure 4 shows an example where the register position can not be changed. Unlike a retiming operation, a resynthesis operation provides the capability of modifying the vertices and connections of a communication graph.

B. Optimization power of retiming and resynthesis

A resynthesis operation itself cannot contribute any changes to the STG of an FSM. However, when combined with retiming, it becomes a handy tool. In essence, the combination of retiming and resynthesis validates the converse of Lemmas 1 and 2 as will be shown in Theorem 1. Moreover, it determines the transitions of newly created states due to forward retiming moves, and thus has decisive effects on initialization sequences as will be discussed in Section V-B. On the other hand, we shall mention an important property about retiming and resynthesis operations.

[5]This commutativity can be understood from the uniqueness of the final communication graph regardless of the order of atomic moves. In other words, for a retime function $\rho = \rho_1 + \cdots + \rho_i$ on a communication graph G, the final register positions are of no differences by applying ρ once to G or by applying $\rho_1, \ldots, \rho_i$ in a sequence of any order to G.

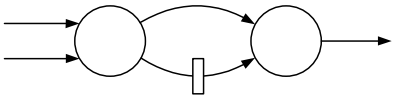

Fig. 4. A subgraph of a communication graph, where the register can be moved neither forwardly nor backwardly.

Lemma 3: Given an FSM, the newly created states (not existing in the original state transition graph) due to atomic moves of retiming remain dangling throughout iterative retiming and resynthesis operations if not merged with the original existing states due to immediate equivalence.

Proof: Prove by induction on the structure of state transition graphs modified by retiming. (Notice that resynthesis is not capable of modifying an STG but is useful in increasing retiming configurations.)

In the base case, there are no newly created states initially. Thus no newly created states can become non-dangling. In the inductive case, assume that, before and at the k-th iteration of retiming (and resynthesis), no newly created dangling states become non-dangling if not merged with the original existing states. Suppose the $(k+1)$-th iteration is performed. Four cases induced by retiming need to be analyzed: state annihilation, creation, merge, and split. However, no dangling states can become non-dangling due to state annihilation and creation. We only need to focus on state merge and split. For state merge, merging two dangling immediately equivalent states yields no non-dangling state because the predecessor states of the new merged state are all dangling. In other words, a state derived from merging two immediately equivalent states is non-dangling only if at least one of its original two states is non-dangling. However, in the inductive hypothesis, we assume that no newly created dangling states become non-dangling before and at the k-th iteration. The non-dangling states must exist in the original state transition graph. Consequently, no dangling states can become non-dangling without merging with the original existing states. For state split, splitting a state q into multiple immediately equivalent states q_1 and q_2 redistributes any incoming edge to q to either q_1 or q_2. As a consequence, if q is dangling, then q_1 and q_2 must be dangling as well because all predecessor states of q (and thus of q_1 and q_2) are dangling. That is, no dangling states can become non-dangling due to state split.

Therefore, the newly created states due to retiming remain dangling throughout iterative retiming and resynthesis operations if not merged with the original existing states. ■

Remark 1: As an orthogonal issue to our discussion on how retiming and resynthesis can alter the STG of an FSM, the transformation of retiming and resynthesis was shown [14] to

have the capability of exploiting various state encodings (or assignments) of an FSM.

Notice that the induced state space of the dangling states originating from atomic moves of retiming is immaterial in our study of the optimization capability of retiming and resynthesis because an FSM after initialization never reaches such dangling states. An exact characterization of the optimization power of retiming and resynthesis is given as follows.

Theorem 1: Ignoring the (unreachable) dangling states created due to retiming, two FSMs are transformable to each other through retiming and resynthesis if, and only if, their state transition graphs are transformable to each other by a sequence of splitting a state into multiple immediately equivalent states and of merging multiple immediately equivalent states into a single state.

Proof: ($\Longrightarrow$) Since resynthesis does not change the transition functions of an FSM, the proof is immediate from Lemmas 1 and 2.

($\Longleftarrow$) Given a target sequence of merging and splitting of immediately equivalent states, it can be accomplished by a sequence of retiming and resynthesis. Essentially, each merging (resp. splitting) of states can be achieved with a resynthesis operation followed by a forward (resp. backward) retiming operation. To see why, let Σ and Q be the input alphabet and state set of $\mathcal{M}$, respectively. Without loss of generality, assume that $q_1, q_2 \in Q$ are immediately equivalent states to be merged. (Merging more than two states can be done similarly.) As illustrated in Figure 5, an resynthesis operation can rewrite the original transition functions $\vec{\delta} : \Sigma \times Q \rightarrow Q$ as a composition of two parts, $\vec{\delta}(\sigma, q) = \vec{\Delta}_2(\sigma, \vec{\Delta}_1(q))$, where $\vec{\Delta}_1 : Q \rightarrow Q\backslash\{q_2\}$ and $\vec{\Delta}_2 : \Sigma \times Q\backslash\{q_2\} \rightarrow Q$. In addition, $\vec{\Delta}_1(q_2) = q_1$, and $\vec{\Delta}_1(q) = q$ for $q \neq q_2$. Retiming registers forward to the positions in-between $\vec{\Delta}_1$ and $\vec{\Delta}_2$ results in a new state transition function $\vec{\Delta}_1 \circ \vec{\Delta}_2 \equiv \vec{\Delta}_1(\vec{\Delta}_2(\sigma, q))$ as shown in Figure 5 (iii). The new transition function in effect merges immediately equivalent states q_1 and q_2. Notice that the retiming operation is always possible because the output functions can be rewritten to depend on $Q\backslash\{q_2\}$ without affecting the global behavior of $\mathcal{M}$.

On the other hand, assume $q' \in Q$ is the state to be split into multiple immediately equivalent states $Q^\dagger$, with $Q^\dagger \cap Q = \emptyset$. As illustrated in Figure 6, an resynthesis operation can again rewrite the original transition functions $\vec{\delta}$ as a composition of two parts, $\vec{\delta} = \vec{\Delta}_4 \circ \vec{\Delta}_3$, where $\vec{\Delta}_3 : \Sigma \times Q \rightarrow Q^\dagger \cup Q\backslash\{q'\}$ and $\vec{\Delta}_4 : Q^\dagger \cup Q\backslash\{q'\} \rightarrow Q$. In addition, $\vec{\Delta}_3(\sigma, q_i) \in Q^\dagger$ for $\vec{\delta}(\sigma, q_i) = q'$, and $\vec{\Delta}_3(\sigma, q_i) = \vec{\delta}(\sigma, q_i)$ for $\vec{\delta}(\sigma, q_i) \neq q'$. Moreover, $\vec{\Delta}_4(q^\dagger) = q'$ for $q^\dagger \in Q^\dagger$ and $\vec{\Delta}_4(q) = q$ for $q \notin Q^\dagger$. Retiming registers to the positions in-between $\vec{\Delta}_3$ and $\vec{\Delta}_4$ results in a new state transition function $\vec{\Delta}_3(\sigma, \vec{\Delta}_4(q))$ as shown in Figure 6 (iii). The new transition function in effect splits q to $Q^\dagger$. Notice that the retiming is always possible because the output functions, originally depending on Q, can be rewritten (by resynthesis) as functions depending on $Q^\dagger \cup Q\backslash\{q\}$.

Consequently, any sequence of merging and splitting of immediately equivalent states is achievable using retiming and resynthesis operations. ■

(A similar result of Theorem 1 appeared in [19], where however the optimization power of retiming and resynthesis was over-stated as will be detailed in Section VI.) Notice that the statement of Theorem 1 is not constructive in the sense that no procedure is given to determine if two FSMs are transformable to each other under retiming and resynthesis. This weakness motivates us to study a constructive alternative.

From Theorem 1, one can show that retiming and resynthesis can not alter the sequential (input-output) behavior of an FSM in the induced state subspace consisting of non-dangling states.

Corollary 1: Given two FSMs $\mathcal{M} = (Q, I, \Sigma, \Omega, \vec{\delta}, \vec{\lambda})$ and $\mathcal{M}^\dagger = (Q^\dagger, I^\dagger, \Sigma, \Omega, \vec{\delta}^\dagger, \vec{\lambda}^\dagger)$, if $\mathcal{M}$ and $\mathcal{M}^\dagger$ are transformable to each other through retiming and resynthesis operations, then there exists a relation $R \subseteq Q_{nd} \times Q^\dagger_{nd}$, where Q_{nd} and $Q^\dagger_{nd}$ are the non-dangling subsets of Q and $Q^\dagger$, respectively, satisfying

1) $\forall (q, q^\dagger) \in R$, $\forall \sigma \in \Sigma . (\vec{\delta}(\sigma, q), \vec{\delta}^\dagger(\sigma, q^\dagger)) \in R$.
2) $\forall (q, q^\dagger) \in R$, $\forall \sigma \in \Sigma . \vec{\lambda}(\sigma, q) = \vec{\lambda}^\dagger(\sigma, q^\dagger)$.

Proof: Since $\mathcal{M}$ and $\mathcal{M}^\dagger$ are transformable to each other through retiming and resynthesis, their state transition graphs for non-dangling states can be transformed to each other by state merging and splitting by Theorem 1. (Because the initial state and thus the reachable states of an implicitly initializable FSM must be non-dangling [17], we can concentrate on non-dangling states.) Without loss of generality, assume $\mathcal{M}$ is the original FSM to be transformed to $\mathcal{M}^\dagger$ through retiming and resynthesis. A relation R satisfying the two criteria can be constructed below. Initially, $\mathcal{M}^\dagger = \mathcal{M}$, let $R = \{(q, q')\}$ for any non-dangling state pair q and q' immediately equivalent in $\mathcal{M}$. (Note that q and q' need not be distinct states). In the following iterative updating, suppose $(q, q_1^\dagger) \in R$ and $q_1^\dagger$ is to be split as $\{q_2^\dagger, q_3^\dagger\}$. Then R is updated by removing $(q, q_1^\dagger)$ and adding $\{(q, q_2^\dagger), (q, q_3^\dagger)\}$. On the other hand, suppose $(q, q_4^\dagger), (q, q_5^\dagger) \in R$ and $q_4^\dagger, q_5^\dagger$ are to be merged as $q_6^\dagger$. Then R is updated by removing $\{(q, q_4^\dagger), (q, q_5^\dagger)\}$ and adding $(q, q_6^\dagger)$. The update process terminates when $\mathcal{M}^\dagger$ is transformed into the final status $(Q^\dagger, I^\dagger, \Sigma, \Omega, \vec{\delta}^\dagger, \vec{\lambda}^\dagger)$. Since the so constructed R satisfies the two criteria along the state merging and splitting transformations, the corollary follows. ■

Since the relation R of Corollary 1 is a strict subset of the general state equivalence relation [11], the input-output behavior of an FSM in the non-dangling state subspace is not affected under retiming and resynthesis.

Remark 2: Peripheral retiming [14] generalizes standard retiming in that edges with negative weights are temporarily allowed. One might ask if this generalization increases the optimization power of retiming and resynthesis. The answer to this question is negative as we argue below.

Peripheral retiming and resynthesis work as follows. A peripheral retiming operation is performed on a communication graph $G = (V, E)$ such that edges with negative weights are allowed to exist temporarily. A resynthesis operation is then performed on the peripheral retimed communication graph, yielding a new communication graph $G^\dagger = (V^\dagger, E^\dagger)$. (To ensure that edges of negative weights will be recovered to possess non-negative weights later, the resynthesis operation needs to preserve these edges in the modified communication graph.) Another retiming operation on $G^\dagger$, yielding $G^\ddagger =$

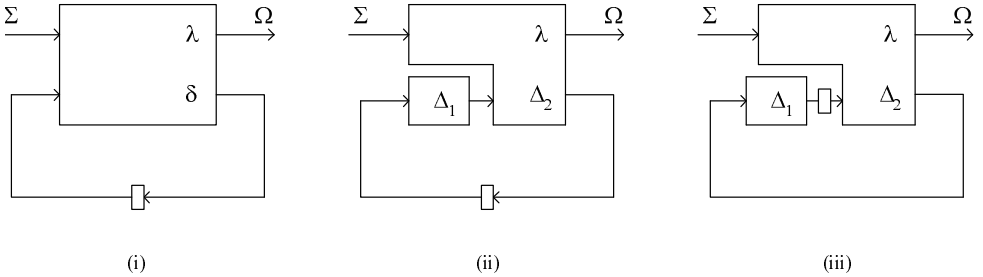

Fig. 5. Given an FSM in (i), it can be resynthesized to the one in (ii), and then forwardly retimed to the one in (iii).

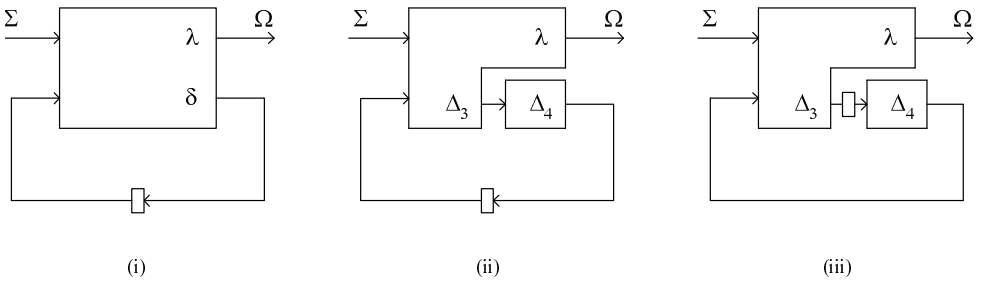

Fig. 6. Given an FSM in (i), it can be resynthesized to the one in (ii), and then backwardly retimed to the one in (iii).

$(V^\dagger, E^\ddagger)$, must ensure that all edges $E^\ddagger$ are of non-negative weights. (If the last step fails, the entire transformation is illegal. We are concerned with legal transformation only.) Observe that the edges with non-zero weights in $E^\dagger$ survive throughout the above operations (as discussed at the end of Section II). That is, these edges exist in both E and $E^\ddagger$ as well, except for some weight changes due to the retiming operations before and after resynthesis. Valuations on state variables of G (respectively $G^\ddagger$) induce valuations on the variables of these edges in G (respectively $G^\ddagger$). Let Q and $Q^\ddagger$ be the state sets of G and $G^\ddagger$, respectively. State pairs $(q \in Q, q^\ddagger \in Q^\ddagger)$ yielding the same valuations on these edges form a state relation of immediate equivalence, similar to the arguments for Lemma 1. Even iterating peripheral retiming and resynthesis cannot provide more transformation power than that specified in Theorem 1. Hence, when combined with resynthesis, peripheral retiming does not provide more transformation power than standard retiming.

It is noteworthy that, although in theory peripheral retiming combined with resynthesis does not increase the transformation power of standard retiming combined with resynthesis, it is useful in practice for design optimization.

C. Retiming-resynthesis equivalence and canonical representation

Given an FSM, the transformation of retiming and resynthesis operations can rewrite it into a class of equivalent FSMs (constrained by Theorem 1). We ask if there exists a computable canonical representative in each such class, and answer this question affirmatively by presenting a procedure constructing it. Rather than arguing directly over FSMs in terms of transition and output functions, we simplify our exposition by arguing over STGs.

Because retiming and resynthesis operations are reversible, we know

Proposition 3: Given STGs G, G_1, and G_2. Suppose G_1 and G_2 are derivable from G using retiming and resynthesis operations. Then G_1 and G_2 are transformable to each other under retiming and resynthesis.

We say that two FSMs (STGs) are *equivalent under retiming and resynthesis* if they are transformable to each other under retiming and resynthesis. Thus, any such equivalence class is *complete* in the sense that any member in the class is transformable to any other member. To derive a canonical representative of any equivalence class, consider the algorithm outlined in Figure 7. Similar to the general state minimization algorithm [8], the idea is to seek a representative minimized with respect to the immediate equivalence of states. However, unlike the least-fixed-point computation of the general state minimization, the computation in Figure 7 looks for a greatest fixed point.[6] Given an STG, the algorithm first removes all the dangling states, and then iteratively merges immediately equivalent states until no more states can be merged.

Theorem 2: Given an STG G, Algorithm *ConstructQuotientGraph* produces a canonical state-minimized solution, which is equivalent to G under retiming and resynthesis.

[6]In the fixed-point computation of the general state minimization, there is initially only one equivalence class, i.e., the universal state set. In the following iterative computation, the state space is refined monotonically, and thus the number of equivalence classes increases monotonically. It can be seen as a least fixed-point computation in the sense that it is analogous to the least fixed-point computation of reachability analysis, where the reached state set increases monotonically. However, unlike the general state minimization, the computation of Figure 7 looks for a greatest fixed point in the following sense. Initially, every equivalence class is a singleton set, consisting of one state. Thus, the number of equivalence classes equals the state size initially. In the iterative computation, equivalence classes are merged with respect to immediate equivalence, and the number of equivalence classes decreases monotonically.

ConstructQuotientGraph
input: a state transition graph G
output: a state-minimized transition graph
w.r.t. immediate equivalence
begin
01 remove dangling states from G
02 **repeat**
03 compute and merge immediately equivalent states
04 **until** no merging performed
05 **return** the reduced graph
end

Fig. 7. Algorithm: Construct quotient graph.

VerifyEquivalenceUnderRetiming&Resynthesis
input: two state transition graphs G_1 and G_2
output: YES, if G_1 and G_2 are equivalent under retiming and resynthesis
NO, otherwise
begin
01 $G_{1/}$:= *ConstructQuotientGraph*(G_1)
02 $G_{2/}$:= *ConstructQuotientGraph*(G_2)
03 **if** $G_{1/}$ and $G_{2/}$ are isomorphic
04 **then return** YES
05 **else return** NO
end

Fig. 8. Algorithm: Verify equivalence under retiming and resynthesis.

Proof: It is clear that the algorithm always terminates for finite state transition graphs.

Recall our assumption that FSMs are of implicit reset. Since dangling states do not affect the normal operation of an FSM (but affect its initialization), the algorithm can safely remove the state space induced by the dangling states and consider only the remaining state space. (See also Proposition 5.)

For the sake of contradiction, assume the algorithm produces two different (non-isomorphic) quotient graphs $G_{1/}$ and $G_{2/}$ for two given STGs G_1 and G_2, respectively, which are equivalent under retiming and resynthesis. Because the algorithm merges only immediately equivalent states, $G_{1/}$ and $G_{2/}$ must also be equivalent under retiming and resynthesis (but not isomorphic by assumption). Since $G_{1/}$ and $G_{2/}$ are not isomorphic, there does not exist a bijection (a one-to-one and onto mapping) between states of $G_{1/}$ and states of $G_{2/}$ such that the bijection preserves immediate equivalence. Two cases need to be considered. First, there exists an onto but not one-to-one mapping from one graph to the other which preserves immediate equivalence. In this case, not both $G_{1/}$ and $G_{2/}$ are maximally reduced. It contradicts with the assumption that any two states in a quotient graph cannot be immediately equivalent. Second, there exists no mapping preserving immediate equivalence. However, from Proposition 3, we know that $G_{1/}$ is transformable to G_1, then to G_2, and finally to $G_{2/}$. Hence a mapping that preserves immediate equivalence must exist between $G_{1/}$ and $G_{2/}$. Again a conflict arises. The theorem follows. ■

For a naïve implementation based on explicit graph enumeration, Algorithm *ConstructQuotientGraph* can be done in time complexity $O(kn^2)$, where k is the size of the input alphabet and n is number of states. This complexity can be obtained from the following analysis. Step 1 of Figure 7 can be done in $O(n^2)$ by iterative removal of states without predecessor states. More specifically, in each iteration, for each state of G, if it has no predecessor states, it is removed from the state transition graph. The process terminates when no more states can be removed. There are at most n iterations, each of time complexity $O(n)$. Therefore, the time complexity for Step 1 is of $O(n^2)$. On the other hand, Steps 2–4 of Figure 7 can be implemented as follows. For $i = 1, \ldots, n-1$, substitute state q_i for q_j ($j > i$) in G if q_i is not substituted before and q_i, q_j are immediately equivalent. To analyze the complexity, there are at most $\sum_{i=1}^{n-1}(n-i)$ comparisons for state pairs q_i and q_j. The time complexity of each comparison is $O(k)$ for checking immediate equivalence. The time complexity for Steps 2–4 of Figure 7 is of $O(k\sum_{i=1}^{n-1}(n-i)) = O(kn^2)$. Therefore, the overall time complexity for Algorithm *ConstructQuotientGraph* is $O(kn^2)$.

Notice that the complexity is exponential when the input is an FSM, instead of an STG, representation. (We distinguish between an FSM, a tuple $(Q, I, \Sigma, \Omega, \vec{\delta}, \vec{\lambda})$, and its STG. For an FSM, its behavior is described with transition and output functions rather than in graph representation. The size (or complexity measure) of an FSM is in terms of the size of binary encodings of its transition and output functions. On the other hand, the size of an STG is in terms of its number of nodes, i.e. states, and edges, i.e. transitions.) For an implicit symbolic implementation, the complexity depends heavily on the internal symbolic representations. If Step 3 in Figure 7 computes and merges all immediately equivalent states at once in a breadth-first-search manner, then the algorithm converges in a minimum number of iterations.

From the proof of Theorem 2, an algorithm outlined in Figure 8 can check if two STGs are transformable to each other under retiming and resynthesis.

Theorem 3: Given two state transition graphs, Algorithm *VerifyEquivalenceUnderRetiming&Resynthesis* verifies if they are equivalent under retiming and resynthesis.

Proof: A direct consequence of Theorem 2. ■

Notice that the algorithm of Figure 8 can be modified to construct retiming and resynthesis steps translating one FSM $\mathcal{M}_1$ to the other $\mathcal{M}_2$. For instance, G_1 of $\mathcal{M}_1$ can be first reduced to the quotient graph, from which we can reverse the reduction procedure of G_2 of $\mathcal{M}_2$ and thus bring G_1 to G_2. As shown in the proof of Theorem 1, retiming and resynthesis operations can be derived from these state manipulations.

Observe the specialty that FSMs are deterministic and with known initial states. Hence the complexity of the algorithm in Figure 8 is the same as that in Figure 7 since the graph isomorphism check for such STGs is $O(kn)$, which is not the dominating factor. With the presented algorithm, checking the equivalence under retiming and resynthesis is not easier than general equivalence checking. In the following section, we investigate its intrinsic complexity.

As an example, Figure 9 shows three STGs (i), (ii), and (iii). Their equivalence under retiming and resynthesis can be checked by Theorem 3. It can be verified that STGs (i) and (ii) are transformable to each other under retiming and resynthesis, but they are not transformable to STG (iii).

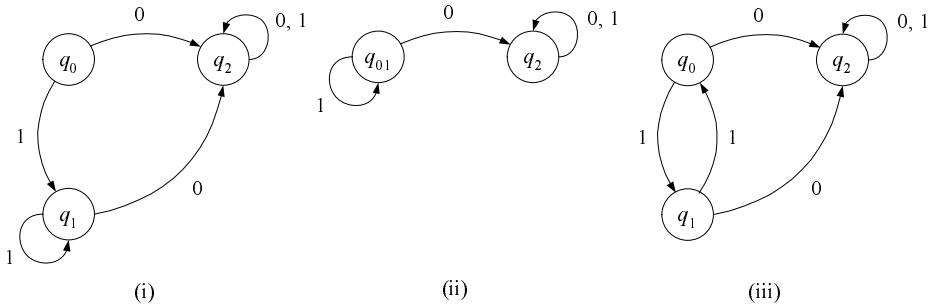

Fig. 9. The STGs in (i) and (ii) are equivalent under retiming and resynthesis transformation. Since states q_0 and q_1 in (i) are immediately equivalent, they can be merged and thus the STG can be simplified to that in (ii). On the other hand, although the STG in (iii) is equivalent to the previous two in terms of input-output behaviors, it is not equivalent to them under retiming and resynthesis transformation.

IV. Verification Complexity

We show some complexity results of verifying if two FSMs are equivalent under retiming and resynthesis.

A. Verification with unknown transformation history

We investigate the complexity of verifying the equivalence of two FSMs with unknown history of (iterative) retiming and resynthesis operations.

Theorem 4: Determining if two FSMs are equivalent under iterative retiming and resynthesis with unknown transformation history is PSPACE-complete.

Proof: Certainly Algorithm *VerifyEquivalenceUnderRetiming&Resynthesis* can be performed in polynomial space (even with inputs in FSM representations).

On the other hand, we need to reduce a PSPACE-complete problem to our problem at hand. The following problem is chosen.

> Given a total function $f : \{1, \ldots, n\} \rightarrow \{1, \ldots, n\}$, is there a composition of f such that, by composing f k times, $f^k(1) = n$?

In other words, the problem asks if n is "reachable" from 1 through f. It was shown [7] to be deterministic[7] LOGSPACE-complete in the unary representation and, thus, PSPACE-complete in the binary representation [16]. We show that the problem in the unary (resp. binary) representation is $\log$-space (resp. polynomial-time) reducible to our problem with inputs in STG (resp. FSM) representations. We further establish that the answer to the PSPACE-complete problem is positive if and only if the answer to the corresponding equivalence verification problem (to be constructed) is negative. Since the complexity class of nondeterministic space is closed under complementation [4], the theorem follows.

To complete the proof, we elaborate the reduction. Given a function f as stated earlier, we construct two total functions $f_1, f_2 : \{0, 1, \ldots, n\} \rightarrow \{0, 1, \ldots, n\}$ as follows. Let f_1 have the same mapping as f over $\{1, \ldots, n-1\}$ and have $f_1(0) = 1$ and $f_1(n) = 1$. Also let f_2 have the same mapping as f with $f_2(0) = 1$ but $f_2(n) = 0$. Clearly the constructions of f_1 and f_2 can be done in log-space. Treating $\{0, 1, \ldots, n\}$ as the state set, functions f_1 and f_2 specify the transitions of two STGs (with an empty input alphabet), say G_1 and G_2, respectively, as shown in Figure 10. In addition, let all the states of G_1 and G_2 have the same output observation. That is, the output functions of the FSMs of G_1 and G_2 do not distinguish states. Under this setting, observe that any state of G_1 (similarly G_2) has exactly one next-state. Thus, every state is either in a single cycle or on a single path leading to a cycle. Observe also that two states of G_1 (similarly G_2) are immediately equivalent if and only if they have the same next-state. An important consequence of these two observations is that any dangling state (not in a cycle) can eventually be merged, due to immediate equivalence, with some non-dangling state (in a cycle) which has the same next state. By Theorem 1, this merging process can be achieved with retiming and resynthesis over the FSMs defined by G_1 and G_2.

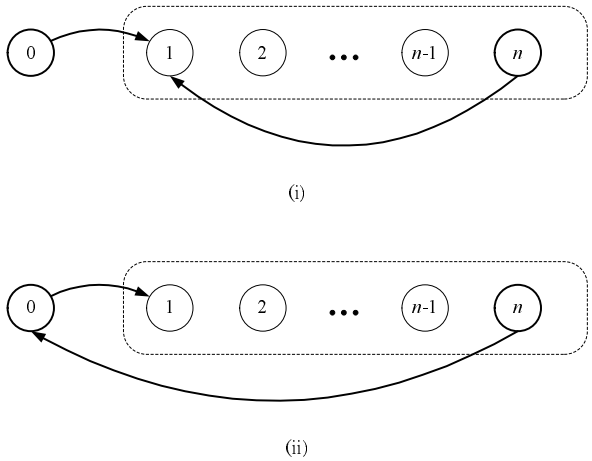

Fig. 10. STGs G_1 in (i) and G_2 in (ii) are induced from functions f_1 and f_2, respectively. These two graphs are identical except for the outgoing edges of node n. The outgoing edges for nodes $\{1, \ldots, n-1\}$ are determined by function f and are omitted.

To see the relationship between reachability and the equivalence under retiming and resynthesis, consider the case where n is reachable from 1 through f. States 1 and n of G_1 must be in a cycle excluding state 0; states 1 and n of G_2 must be in a cycle including state 0. Hence the state-minimized (with respect to immediate equivalence) graphs of G_1 and G_2

[7]It is a well-known result by Savitch [20] that deterministic and nondeterministic space complexities coincide.

are not isomorphic. That is, G_1 and G_2 are not equivalent under retiming and resynthesis. On the other hand, consider the case where n is unreachable from 1 through f. Then state n of G_1 and state n of G_2 are dangling. From the mentioned observations, merging dangling states with non-dangling states in G_1 and in G_2 yields two isomorphic graphs. (The isomorphism can be established by a mapping π from the set of non-dangling states of G_1 to that of G_2, and vice versa, with $\pi(i) = i$.) That is, G_1 and G_2 are equivalent under retiming and resynthesis. Therefore, n is reachable from 1 through f if, and only if, G_1 and G_2 are not equivalent under retiming and resynthesis. (Notice that, unlike the discussion of optimization capability, here we should not ignore the effects of retiming and resynthesis over the unreachable state space.) ■

B. Verification with known transformation history

By Theorem 4, verifying if two FSMs are equivalent under retiming and resynthesis without knowing the transformation history is as hard as the general equivalence checking problem. Thus, we advocate a conservative design methodology optimizing synchronous hardware systems to ameliorate verifiability.

An easy approach to circumvent the PSPACE-completeness is to record the history of retiming and resynthesis operations as verification checkpoints, or alternatively to perform equivalence checking after every retiming or resynthesis operation. The reduction in complexity results from the following well-known facts.

Proposition 4: Given two synchronous hardware systems, verifying if they are transformable to each other with retiming is of the same complexity as checking graph isomorphism (for communication graphs without edge weights), which is within NP ∩ coNP; verifying if they are transformable to each other with resynthesis is of the same complexity as combinational equivalence checking, which is coNP-complete.

Therefore, if transformation history is completely known, the verification complexity reduces to coNP-complete.

V. Initialization Sequences

To discuss initialization sequences, we rely on the following proposition of Pixley [17].

Proposition 5: ([17]) The initial states of an initializable FSM can not be dangling. Moreover, any non-dangling state of an initializable FSM can be used as an initial state by suitably modifying initialization sequences.

By Corollary 1, the behavior of an FSM in non-dangling states cannot be altered by retiming and resynthesis. Also, by Lemma 3, newly created states by retiming (and resynthesis) not immediately equivalent to any non-dangling states remain dangling throughout iterative retiming and resynthesis operations. Adding dangling states does not affect initializability because prefixing an original initialization sequence with a long enough input sequence can drive an FSM to some non-dangling state, which is a legitimate initial state by Proposition 5. (Note that any dangling state will eventually reach some non-dangling state after a long enough input sequence is applied, regardless of the input patterns.) As a result,

Corollary 2: The initializability of an FSM is an invariant under retiming and resynthesis.

Hence we shall assume that the given FSM $\mathcal{M}$ is initializable. Furthermore, we assume that its initialization sequence is given as a black box. That is, we have no knowledge on how $\mathcal{M}$ is initialized. Under these assumptions, we study how the initialization sequence is affected when $\mathcal{M}$ is retimed (and resynthesized). As shown earlier, the creation and annihilation of dangling states are immaterial to the optimization capability of retiming and resynthesis. However, they play a decisive role in affecting initialization sequences. In essence, the longest transition path among dangling states determines how long the initialization sequences should be increased.

A. Initialization affected by retiming

Lag-dependent bounds. Effects of retiming on initialization sequences were studied by Leiserson and Saxe in [9], where their *Retiming Lemma* can be rephrased as follows.

Lemma 4: ([9]) Given a communication graph $G = (V, E)$ and a normalized retime function ρ, let $\ell = \max_{v \in V} -\rho(v)$ and let $G^\dagger$ be the corresponding retimed communication graph of G. Suppose $\mathcal{M}$ and $\mathcal{M}^\dagger$ are the FSMs specified by G and $G^\dagger$, respectively. Then after $\mathcal{M}^\dagger$ is initialized with an arbitrary input sequence of length ℓ, any state of $\mathcal{M}^\dagger$ has an equivalent[8] state in $\mathcal{M}$.

That is, prefixing the original initialization sequence of $\mathcal{M}$ with an arbitrary input sequence of length no less than ℓ results in a valid initialization sequence for $\mathcal{M}^\dagger$. Thus, ℓ (nonnegative for normalized ρ)[9] gives an upper bound of the increase of initialization sequences under retiming. This bound was further tightened in [2], [22] by letting ℓ be the maximum of $-\rho(v)$ for all v of functional elements whose functions define non-surjective mappings. Unfortunately, this strengthening still does not produce an exact bound. Moreover, by Proposition 1, a normalized retime function among its equivalent retime functions may not be the one that gives the tightest bound. A derivation of exact bounds will be discussed in Section V-B.

Lag-independent bounds. Given a synchronous hardware system, a natural question is if there exists some bound which is universally true for all possible retiming operations. Even though the bound may be looser than lag-dependent bounds, it discharges the construction of new initialization sequences from knowing what retime functions have been applied. Indeed, such a bound does exist as exemplified below.

Proposition 6: Given a communication graph $G = (V, E)$ and a normalized retime function ρ, let $r(v)$ denote the minimum number of registers along any path from the host to vertex v. Then $r(v)$ sets an upper bound of the number of registers that can be moved forward across v, i.e., $r(v) \geq -\rho(v)$. (Note that $\rho(v)$ is negative for forward retiming.)

[8] A state q of FSM $\mathcal{M}$ is equivalent to a state $q^\dagger$ of FSM $\mathcal{M}^\dagger$ if $\mathcal{M}$ starting from q, and $\mathcal{M}^\dagger$ starting from $q^\dagger$ have the same input-output behavior.

[9] Recall that a normalized retime function ρ is with $\rho(\text{host}) = 0$.

Similarly, $r(v)$ on G with reversed edges sets an upper bound of $\rho(v)$.

Thus, $\max_v r(v)$, which is intrinsic to a communication graph and is independent of retiming operations, yields a lag-independent bound.

When initialization delay is not a concern for a synchronous system, one can even relax the above lag-independent bound by saying that the total number of registers of the system is another lag-independent bound. As an example, suppose a system has one million registers and its retimed version runs at one gigahertz clock frequency. Then the initialization delay increased due to retiming is less than a thousandth of a second.

B. Initialization affected by retiming and resynthesis

So far we have focused on initialization issues arising when a system is retimed only. Here we extend our study to issues arising when a system is iteratively retimed and resynthesized.

A difficulty emerges from directly applying Lemma 4 to bound the increase of initialization sequences under iterative retiming and resynthesis. Interleaving retiming with resynthesis makes the union bound $\sum_i u_i$ the only available bound from Lemma 4, where u_i denotes the lag-dependent bound for the ith retiming operation. Essentially, inaccuracies accumulate along with the summation of the union bound. Thus, the bound derived this way can be far beyond what is necessary. In the light of lag-independent bounds discussed earlier, one might hope that there may exist some constant which upper bounds the increase of initialization sequences due to any iterative retiming and resynthesis operations. (Notice that, when no resynthesis operation is performed, the transformation of a series of retiming operations can be achieved by a single retiming operation. Thus a lag-independent bound exists for *iterative* retiming operations.) Unfortunately, such a transformation-independent bound does not exist as shown in Theorem 5.

Lemma 5: Any dangling state of an FSM (with implicit reset) is removable through iterative retiming and resynthesis operations.

Proof: By Proposition 5, the initial states of an FSM $\mathcal{M}$ with implicit reset must be non-dangling. Removing dangling states cannot affect the behavior of $\mathcal{M}$. Essentially, states without predecessor states can be eliminated with a resynthesis operation followed by a retiming operation. To see why this is the case, let Σ be the input alphabet, Q be the set of states of $\mathcal{M}$, and $Q^\dagger \subseteq Q$ be the subset of states with predecessors. As illustrated in Figure 11, a resynthesis operation can rewrite the original transition functions $\vec{\delta} : \Sigma \times Q \rightarrow Q$ as a composition of three parts $\vec{\delta} = \vec{\Delta}^{-1} \circ \vec{\Delta} \circ \vec{\delta}$, where $\vec{\Delta} : Q \rightarrow Q^\dagger$, $\vec{\Delta}^{-1} : Q^\dagger \rightarrow Q$, and $\vec{\Delta}^{-1} \circ \vec{\Delta}$ is an identity mapping. (Notice that $\vec{\Delta}^{-1}$ exists because states $Q \backslash Q^\dagger$ have empty pre-image.) Retiming registers backward to the positions in-between $\vec{\Delta}$ and $\vec{\Delta}^{-1}$ eliminates states with no predecessors. (The retiming operation is possible because the output functions of $\mathcal{M}$ can take the intermediate valuation after $\vec{\delta}$ and before the identity mapping $\vec{\Delta}^{-1} \circ \vec{\Delta}$ as its state input.) Therefore, with iterative retiming and resynthesis, dangling states are removable. ■

Theorem 5: Given a synchronous hardware system and an arbitrary constant c, there always exist retiming and resynthesis operations on the system such that the length increase of the initialization sequence exceeds c.

Proof: Any dangling state of an FSM can be removed by iterative retiming and resynthesis by Lemma 5. On the other hand, since the transformation of retiming and resynthesis is reversible, a path over dangling states can be made arbitrary long through iterative retiming and resynthesis operations. Therefore, the theorem follows. ■

Since the mentioned union bound is inaccurate and requires knowing the applied retime functions, it motivates us to investigate the computation of exact[10] length increase of initialization sequences without knowing the history of retiming and resynthesis operations. The length increase can be derived by computing the length, say n, of the longest transition paths among the dangling states because applying an arbitrary[11] input sequence of length greater than n drives the system to a non-dangling state. The length n can be obtained using a symbolic computation. By breadth-first search, one can iteratively remove states without predecessor states until a greatest fixed point is reached. The number of the performed iterations is exactly n.

VI. Related Work

Optimization capability. The closest to our work on the optimization power of retiming and resynthesis is [19], where the optimization power was unfortunately over-stated contrary to the claimed exactness. The mistake resulted from the claim that any *2-way switch* (redirecting a transition to another immediately equivalent next state) operation is achievable using *2-way merge* (merging two immediately equivalent states into a single state) and *2-way split* (splitting a state into two immediately equivalent states) operations, see [19] for detailed illustrations. Figure 12 shows a counterexample illustrating a 2-way switch operation that is not achievable with 2-way merge and split operations. The over-stated optimization power results from the overlooked fact that, under any input assignment, the next states of immediately equivalent states split from a current state must be the same. In fact, only 2-way merge and split operations are essential. Aside from this minor error, no constructive algorithm was known to determine if two given FSMs are equivalent under retiming and resynthesis. In addition, not discussed were the creation and annihilation of dangling states, which we show to be crucial in initializing synchronous hardware systems.

Verification complexity. Ranjan in [18] examined a few verification complexities for cases under one retiming operation and up to two resynthesis operations with unknown transformation history. The complexity for the case under an arbitrary number of iterative retiming and resynthesis operations was left open,

[10] The exactness is true under the assumption that the initialization sequence of the original FSM is given as a block box. If the initialization mechanism is explored, more accurate analysis may be achieved.

[11] Although exploiting some particular input sequence may shorten the length increase, it complicates the computation.

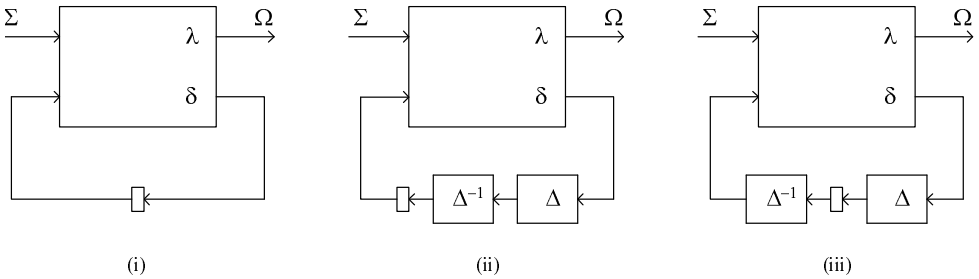

Fig. 11. Given an FSM in (i), it can be resynthesized to the one in (ii), and then backwardly retimed to the one in (iii).

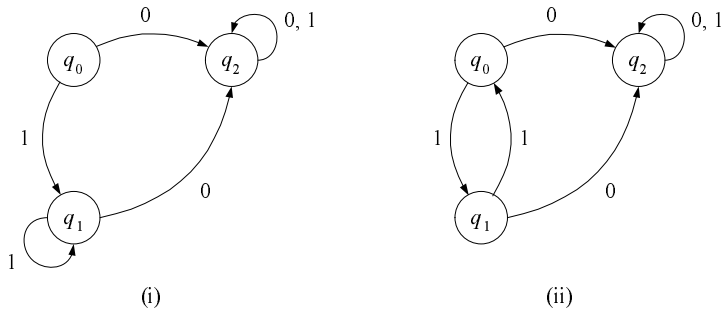

Fig. 12. The STG in (i) is transformable to the STG in (ii) by a 2-way switch operation while the reverse direction is not transformable. Since the operation is not reversible, it falls beyond the transformation power of retiming and resynthesis. In these two STGs, only input labels are shown while output labels are omitted.

and was conjectured in [25] to be easier than the general equivalence checking problem. We disprove the conjecture.

Initialization sequences. For systems with explicit reset, the effect of retiming on initial states was studied in [24], [3], [21]. In the explicit reset case, incorporating resynthesis with retiming does not contribute additional difficulty. Note that, for systems with explicit-reset registers, forward moves of retiming are preferable to backward moves in maintaining equivalent initial states, contrary to the case for systems with implicit-reset registers. To prevent backward moves, Even et al. in [3] proposed an algorithm to find a retime function such that the maximum lag among all vertices is minimized. Interestingly enough, their algorithm can be easily modified to obtain minimum lag-dependent bounds on the increase of initialization sequences (by avoiding forward retiming instead of backward retiming). As mentioned earlier, explicit reset can be seen as a special case of implicit reset when reset circuitry is explicitly represented in the communication graph. Hence, the study of the implicit reset case is more general, and is subtler when considering resynthesis in addition to retiming.

Pixley in [17] studied the initialization of synchronous hardware systems with implicit reset in a general context. Leiserson and Saxe studied the effect of retiming on initialization sequences in [9], where a lag-dependent bound was obtained and was later improved by [2], [22]. We show a lag-independent bound instead. In recent work [15], a different approach was taken to tackle the initialization issue raised by retiming. Rather than increasing initialization sequence lengths, a retimed system was further modified to preserve its original initialization sequence. This modification might need to pay area/performance penalties and could nullify the gains of retiming operations. In addition, the modification requires expensive computation involving existential quantification, which limits the scalability of the approach to large systems. In comparison, prefixing the original initialization sequence with an arbitrary input sequence of a certain length provides a much simpler solution (without modifying the system) to the initialization problem.

On the other hand, we extend our study to the unexplored case of iterative retiming and resynthesis, and show the unboundability of the increase of initialization sequences. Finally, our exact analysis on the increase of initialization sequences is applicable to the case of iterative retiming and resynthesis and improves the bound of [2], [22].

VII. Conclusions and Open Problems

This paper demonstrated some transformation invariants under retiming and resynthesis. Three main results about retiming and resynthesis were established. First, an algorithm was presented to construct a canonical representative of an equivalence class of FSMs transformed under retiming and resynthesis. It was extended to determine if two FSMs are transformable to each other under retiming and resynthesis. Second, a PSPACE-complete complexity was proved for the above problem when the transformation history of retiming and resynthesis is unknown. Hence, to reduce complexity (from PSPACE-complete to coNP-complete), it is indispensable to maintain transformation history, or to check intermediate equivalence after every retiming or resynthesis operation. Third, the effects of retiming and resynthesis on initialization sequences were studied. A lag-independent bound was shown on the length

increase of initialization sequences of FSMs under retiming; in contrast, unboundability was shown on the case under retiming and resynthesis. In addition, an exact analysis on the length increase was presented. We believe our results may reveal some directions enhancing the practicality of retiming and resynthesis for the optimization of synchronous hardware systems.

For future work, it is important to investigate more efficient computation, with reasonable accuracy, for the length increase of initialization sequences for FSMs transformed under retiming and resynthesis. On the other hand, it may seem that our lag-independent bound can be used to improve retiming algorithms by pruning out spurious linear constraints, similar to [12]. Moreover, as the result of [3] can be modified to obtain a retime function targeting area optimization with minimum increase of initialization sequences as discussed in Section VI, it would be useful to study retiming under other objectives while avoiding increasing initialization sequences.

While verifying the equivalence of two sequential circuits transformed by an unbounded number of retiming and resynthesis iterations was shown to be PSPACE-complete, it is open when the number is bounded by some constant. In particular, it is not known if the complexities parametric upon this constant follow the polynomial-time hierarchy [23].

REFERENCES

[1] G. De Micheli. Synchronous logic synthesis: algorithms for cycle-time minimization. *IEEE Trans. on Computer-Aided Design*, vol. 10, pp.63–73, Jan. 1991.

[2] A. El-Maleh, T. E. Marchok, J. Rajski, and W. Maly. Behavior and testability preservation under the retiming transformation. *IEEE Trans. on Computer-Aided Design*, vol. 16, pp.528–543, May 1997.

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

[4] N. Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*. vol. 17, pp.935–938, 1988.

[5] J.-H. Jiang. On some transformation invariants under retiming and resynthesis. In *Proc. Int'l Conf. on Tools and Algorithms for the construction and analysis of systems*, pp.413–428, 2005.

[6] J.-H. Jiang and R. Brayton. On the verification of sequential equivalence. *IEEE Trans. on Computer-Aided Design*, vol. 6, pages 686–697, June 2003.

[7] N. Jones. Space-bounded reducibility among combinatorial problems. *Journal of Computer and System Sciences*, vol. 11, pp.68–85, 1975.

[8] Z. Kohavi, *Switching and Finite Automata Theory*. McGraw-Hill, 1978.

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

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

[11] B. Lin, H. J. Touati, and A. R. Newton. Don't care minimization of multi-level sequential logic networks. In *Proc. Int'l Conf. on Computer-Aided Design*, pages 414–417, 1990.

[12] N. Maheshwari and S. Sapatnekar. Efficient retiming of large circuits. *IEEE Trans. on Very Large Scale Integration Systems*, vol. 6, pages 74–83, March 1998.

[13] S. Malik. *Combinational Logic Optimization Techniques in Sequential Logic Synthesis*. PhD thesis, University of California, Berkeley, 1990.

[14] S. Malik, E. M. Sentovich, R. K. Brayton, A. Sangiovanni-Vincentelli. Retiming and resynthesis: optimization of sequential networks with combinational techniques. *IEEE Trans. Computer-Aided Design*, pp.74–84, Jan. 1991.

[15] M. N. Mneimneh, K. A. Sakallah, and J. Moondanos. Preserving synchronizing sequences of sequential circuits after retiming. In *Proc. Asia and South Pacific Design Automation Conference*, Jan. 2004.

[16] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[17] C. Pixley. A theory and implementation of sequential hardware equivalence. *IEEE Trans. Computer-Aided Design*, vol. 11, pp.1469–1478, Dec. 1992.

[18] R. K. Ranjan. *Design and Implementation Verification of Finite State Systems*. Ph.D. thesis, University of California at Berkeley, 1997.

[19] R. K. Ranjan, V. Singhal, F. Somenzi, and R. K. Brayton. On the optimization power of retiming and resynthesis transformations. In *Proc. Int'l Conf. on Computer-Aided Design*, pp.402–407, Nov. 1998.

[20] W. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, pp.177–192, 1970.

[21] V. Singhal, S. Malik, and R. K. Brayton. The case for retiming with explicit reset circuitry. In *Proc. Int'l Conf. on Computer-Aided Design*, pp.618–625, 1996.

[22] V. Singhal, C. Pixley, R. L. Rudell, and R. K. Brayton. The validity of retiming sequential circuits. In *Proc. Design Automation Conference*, pp.316–321, 1995.

[23] L. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, vol. 3, pp.1–22, 1977.

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

[25] H. Zhou, V. Singhal, and A. Aziz. How powerful is retiming? In *Proc. Int'l Workshop on Logic Synthesis*, 1998.

PLACE PHOTO HERE

Jie-Hong R. Jiang received the B.S. and M.S. degrees in Electronics Engineering from National Chiao Tung University, Hsinchu, Taiwan, in 1996 and 1998, respectively, and the Ph.D. degree in Electrical Engineering and Computer Sciences from the University of California at Berkeley in 2004.

During his compulsory military service, from 1998 to 2000, he was a Second Lieutenant with the Air Force, R.O.C. He was a postdoctoral researcher at the University of California, Berkeley, until joining the Department of Electrical Engineering of National Taiwan University as an assistant professor in August 2005. His current research interests are in the analysis and optimization of sequential systems.

Dr. Jiang is a member of Phi Tau Phi, IEEE, and ACM.

PLACE PHOTO HERE

Robert K. Brayton received the B.S.E.E. degree from Iowa State University in 1956, and the Ph.D. degree in Mathematics from the Massachusetts Institute of Technology in 1961. From 1961 to 1987 he was a member of the Mathematical Sciences Department of the IBM T. J. Watson Research Center, Yorktown Heights, NY. In 1987, he joined the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, where he is the Cadence Distinguished Professor of Engineering.

Prof. Brayton is a member of the National Academy of Engineering, and a Fellow of the IEEE and the American Association for the Advancement of Science. He received the following awards: the IEEE Circuits and Systems Technical Achievement Award (1991), the IEEE Guilleman-Cauer Award (1971), the ISCAS Darlington Award (1987), the Circuits and Systems Golden Jubilee Medal (2000), the IEEE Millennium Medal (2000), the Iowa State Marston Medal (2002), the IEEE Emanuel R. Piore award (2006), and the European Design Automation Association (EDAA) lifetime achievement award (2006).