On Some Transformation Invariants Under Retiming and Resynthesis*

Jie-Hong R. Jiang

Department of Electrical Engineering and Computer Sciences,
University of California, Berkeley

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, established is a lag-independent bound 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. Fortunately, an algorithm computing the exact length increase is presented.

1 Introduction

Retiming [7, 8] 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

* This work was supported in part by NSF grant CCR-0312676, the California Micro program, and our industrial sponsors, Fujitsu, Intel, Magma and Synplicity.

N. Halbwachs and L. Zuck (Eds.): TACAS 2005, LNCS 3440, pp. 413–428, 2005.

the optimization capability is somewhat limited. Therefore, resynthesis [9, 1, 10] 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 [9], e.g. [14, 15, 20]. 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 3) Characterize constructively the transformation power of retiming and resynthesis.
- (Section 4) Prove the PSPACE-completeness of verifying the equivalence of systems transformed by retiming and resynthesis operations when the transformation history is lost.
- (Section 5) Demonstrate the effects of retiming and resynthesis on the initialization sequences of synchronous systems. Present an algorithm correcting initialization sequences.

2 Preliminaries

In this paper, to avoid later complication 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 $\mathcal{V}_2$. Suppose s is a (current-)state variable. Its primed version s' denotes the corresponding next-state variable.

Synchronous Hardware Systems. Based on [7], 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 $\boldsymbol{f}$ if the functional element it represents is of function $\boldsymbol{f}$ (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 non-negative 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*[1]) by applying some input sequences, the so-called *initialization* (or *reset*) *sequences* [13]. It turns out that explicit-reset registers can be replaced with implicit-reset ones plus some reset circuitry [10, 17]. (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 $(Q, I, \Sigma, \Omega, \boldsymbol{\delta}, \boldsymbol{\lambda})$, where Q is a finite set of states, $I \subseteq Q$ is the set of initial states, Σ and Ω are the sets of input and output alphabets, respectively, and $\boldsymbol{\delta} : \Sigma \times Q \to Q$ (resp. $\boldsymbol{\lambda} : \Sigma \times Q \to \Omega$) is the transition function (resp. output function). Let $\mathcal{V}_S$, $\mathcal{V}_I$, and $\mathcal{V}_O$ be the sets of variables that encode the states, input alphabets, and output alphabets respectively. Then $Q = [\![\mathcal{V}_S]\!]$, $\Sigma = [\![\mathcal{V}_I]\!]$ and $\Omega = [\![\mathcal{V}_O]\!]$. To uniquely construct an FSM from a communication graph $G = (V, E)$, we divide each edge $(u, v)\langle w\rangle \in E$ into $w+1$ edges separated by w registers and connected with the two end-vertices u and v. We then associate the outgoing (incoming) edges of registers with current-state variables $\mathcal{V}_S$ (next-state variables $\mathcal{V}_{S'}$); associate the outgoing (incoming) edges of the host with variables $\mathcal{V}_I$ ($\mathcal{V}_O$). All other edges are associated with internal variables. The transition and output functions

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

are obtained, starting from $\mathcal{V}_{S'}$ and $\mathcal{V}_O$, respectively, by a sequence of recursive substitutions of variables with functions of their input functional elements until the functions depend only on variables $\mathcal{V}_I \cup \mathcal{V}_S$.

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

Definition 1. *Given an FSM $\mathcal{M} = (Q, I, \Sigma, \Omega, \boldsymbol{\delta}, \boldsymbol{\lambda})$, two states $q_1, q_2 \in Q$ are* **immediately equivalent** *if $\boldsymbol{\delta}(\sigma, q_1) \equiv \boldsymbol{\delta}(\sigma, q_2)$ and $\boldsymbol{\lambda}(\sigma, q_1) \equiv \boldsymbol{\lambda}(\sigma, q_2)$ for any $\sigma \in \Sigma$.*

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**.

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 not our focus, the exposition of retiming in the optimization perspective is omitted in this paper. Interested readers are referred to [8].) Formally speaking, retiming can be described with a *retime function* [7] 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 $\rho(\text{host}) \equiv 0$, 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, $(u, v)\langle w + \rho(v) - \rho(u) \rangle \in E^\dagger$. It is immediate that the retime function $-\rho$ reverses the retiming from $G^\dagger$ to G.

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 with respect to a communication graph, offsetting ρ by an integer constant c results in an equivalent retime function.*

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.)

3 Optimization Capability

The transformation power of retiming and resynthesis can be understood best with state transition graphs (STGs) defined by FSMs. We investigate how retiming and resynthesis operations can alter STGs.

3.1 Optimization Power of Retiming

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

To study the effect of an atomic backward move, consider a normalized retime function ρ with $\rho(v) = 1$ for some vertex $v \in V$ 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 $\mathcal{V}_S = \mathcal{V}_{S^\natural} \cup \mathcal{V}_{S^*}$ be the state variables of $\mathcal{M}$, where $\mathcal{V}_{S^\natural} = \{s_1, \ldots, s_i\}$ and $\mathcal{V}_{S^*} = \{s_{i+1}, \ldots, s_n\}$ are disjoint. Suppose v is of type $\boldsymbol{f} : \llbracket \{t_1, \ldots, t_j\} \rrbracket \to \llbracket \{s'_1, \ldots, s'_i\} \rrbracket$, where the valuation of next-state variables s'_k is defined by $f_k(t_1, \ldots, t_j)$ for $k = 1, \ldots, i$. Let $\mathcal{M}^\dagger = (\llbracket \mathcal{V}_S^\dagger \rrbracket, I^\dagger, \Sigma, \Omega, \boldsymbol{\delta}^\dagger, \boldsymbol{\lambda}^\dagger)$ be 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 $q_1^\dagger[\mathcal{V}_{S^*}] \equiv q_2^\dagger[\mathcal{V}_{S^*}]$ and $\boldsymbol{f}(q_1^\dagger[\mathcal{V}_T]) \equiv \boldsymbol{f}(q_2^\dagger[\mathcal{V}_T])$, then $q_1^\dagger$ and $q_2^\dagger$ are immediately equivalent. This immediate equivalence results from the fact that the transition and output functions of $\mathcal{M}^\dagger$ can be valuated after the valuation of $\boldsymbol{f}$, which filters out the difference between $q_1^\dagger$ and $q_2^\dagger$. Comparing state pairs between $\mathcal{M}$ and $\mathcal{M}^\dagger$, we can always find a relation $R \subseteq \llbracket \mathcal{V}_S \rrbracket \times \llbracket \mathcal{V}_S^\dagger \rrbracket$ such that

1. Pairs $(q_1, q_1^\dagger)$ and $(q_1, q_2^\dagger)$ are both in R for the state q_1 of $\mathcal{M}$ with $q_1[\mathcal{V}_{S^*}] \equiv q_1^\dagger[\mathcal{V}_{S^*}]$ and $q_1[\mathcal{V}_{S^\natural}] \equiv \boldsymbol{f}(q_1^\dagger[\mathcal{V}_T])$.
2. It preserves the immediate equivalence, that is, $(q, q^\dagger) \in R$ if, and only if, $\boldsymbol{\lambda}(\sigma, q) \equiv \boldsymbol{\lambda}^\dagger(\sigma, q^\dagger)$ and $(\boldsymbol{\delta}(\sigma, q), \boldsymbol{\delta}^\dagger(\sigma, q^\dagger)) \in R$ for any $\sigma \in \Sigma$.

Since $\boldsymbol{f}$ 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 5.) On the other hand, since $\boldsymbol{f}$ may not be a surjective (or an onto) mapping in general, there may be some state q of $\mathcal{M}$ such that $\forall x \in [\![\mathcal{V}_T]\!]$. $q[\mathcal{V}_{S^\natural}] \not\equiv \boldsymbol{f}(x)$, 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 [15], where the phenomena of state creation and annihilation were omitted.)

Note 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 $m_{i+1}, \ldots, m_n$. Clearly all the newly created states not merged with original existing states due to immediate equivalence are dangling. However, to be shown in Section 5.1, the transition paths among these dangling states cannot be arbitrarily long.

Since a retiming operation consists of a series of 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, e.g., the converses of Lemmas 1 and 2 are not true. That is, there may not exist atomic moves of retiming (over a communication graph) which meet arbitrary targeting changes on an STG. Unlike a retiming operation, a resynthesis operation provides the capability of modifying the vertices and connections of a communication graph.

3.2 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 5.2. On the other hand, we shall mention an important property about retiming and resynthesis operations.

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

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 [10] to have the capability of exploiting various state encodings (or assignments) of the 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.*

(A similar result of Theorem 1 appeared in [15], where however the optimization power of retiming and resynthesis was over-stated as will be detailed in Section 6.) From Theorem 1, one can relate two FSMs before and after the transformation of retiming and resynthesis as follows.

Corollary 1. *Given* $\mathcal{M} = (Q, I, \Sigma, \Omega, \boldsymbol{\delta}, \boldsymbol{\lambda})$ *and* $\mathcal{M}^\dagger = (Q^\dagger, I^\dagger, \Sigma, \Omega, \boldsymbol{\delta}^\dagger, \boldsymbol{\lambda}^\dagger)$, *FSMs* $\mathcal{M}$ *and* $\mathcal{M}^\dagger$ *are transformable to each other through retiming and resynthesis operations if, and only if, there exists a relation* $R \subseteq Q \times Q^\dagger$ *satisfying*

1. *Any non-dangling state* $q \in Q$ *(resp.* $q^\dagger \in Q^\dagger$*) has at least one non-dangling state* $q^\dagger \in Q^\dagger$ *(resp.* $q \in Q$*) such that* $(q, q^\dagger) \in R$.
2. *State pair* $(q, q^\dagger) \in R$ *if and only if, for any* $\sigma \in \Sigma$, $\boldsymbol{\lambda}(\sigma, q) \equiv \boldsymbol{\lambda}^\dagger(\sigma, q^\dagger)$ *and* $(\boldsymbol{\delta}(\sigma, q), \boldsymbol{\delta}^\dagger(\sigma, q^\dagger)) \in R$.

Notice that the statements of Theorem 1 and Corollary 1 are nonconstructive 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.

3.3 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 Corollary 1). We ask

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 of G
04 **until** no merging performed
05 **return** the reduced graph
end

Fig. 1. Algorithm: Construct quotient graph

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, 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 1. Similar to the general state minimization algorithm [6], 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 1 looks for a greatest fixed point. 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.*

For a naïve explicit enumerative implementation, Algorithm *ConstructQuotientGraph* is of time complexity $O(kn^3)$, where k is the size of input alphabet and n is number of states. (Notice that the complexity is exponential when the input is an FSM, instead of an STG, representation.) For an implicit symbolic implementation, the complexity depends heavily on the internal symbolic representations. If Step 3 in Figure 1 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.

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. 2. Algorithm: Verify equivalence under retiming and resynthesis

An algorithm outlined in Figure 2 can check if two STGs are transformable to each other under retiming and resynthesis.

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

The complexity of the algorithm in Figure 2 is the same as that in Figure 1 since the graph isomorphism check for 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.

4 Verification Complexity

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

4.1 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 [5] to be deterministic[2] LOGSPACE-complete in the unary representation and, thus, PSPACE-complete in the binary representation [12]. 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, f_1 and f_2 specify the transitions of two STGs, say G_1 and G_2, (which have empty input and output alphabets). 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 observations is that all states not in cycles can be merged through iterative retiming and resynthesis due to immediate equivalence.

To see the relationship between reachability and 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 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 in G_1 and G_2 yields two isomorphic graphs. 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.) ■

4.2 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.

[2] It is a well-known result by Savitch [16] that deterministic and nondeterministic space complexities coincide.

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; verifying if they are transformable to each other with resynthesis is of the same complexity as combinational equivalence checking, which is NP-complete.*

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

5 Initialization Sequences

To discuss initialization sequences, we rely on the following proposition of Pixley [13].

Proposition 5. ([13]) *An FSM is initializable only if its initial states are non-dangling. (In fact, any non-dangling state can be used as an initial state by suitably modifying initialization sequences.)*

By Lemma 3, Corollary 1 and Proposition 5, it is immediate that

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.

5.1 Initialization Affected by Retiming

Lag-dependent bounds. Effects of retiming on initialization sequences were studied by Leiserson and Saxe in [7], where their *Retiming Lemma* can be rephrased as follows.

Lemma 4. ([7]) *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[3] state in $\mathcal{M}$.

That is, ℓ (nonnegative for normalized ρ) gives an upper bound of the increase of initialization sequences under retiming. This bound was further tightened in [2, 18] 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 5.2.

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., $-r(v) \leq \rho(v)$. (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.

5.2 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

[3] 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.

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.*

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.*

Since the mentioned union bound is inaccurate and requires knowing the applied retime functions, it motivates us to investigate the computation of exact[4] 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[5] 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.

6 Related Work

Optimization Capability. The closest to our work on the optimization power of retiming and resynthesis is [15], where the optimization power was unfortunately over-stated contrary to the claimed exactness. The mistake resulted from the claim that any *2-way switch* operation is achievable using *2-way merge* and *2-way split* operations (see [15] for their definitions). (Essentially, a restriction needs to be imposed — under any input assignment, the next state of a current state to be split should be unique.) 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.

[4] 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.

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

Verification Complexity. Ranjan in [14] 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, and was conjectured in [20] 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 [19, 3, 17]. 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. 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 [13] 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 [7], where a lag-dependent bound was obtained and was later improved by [2, 18]. We show a lag-independent bound instead. In recent work [11], 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 an arbitrary input sequence of a certain length to the original initialization sequence provides a much simpler solution (without modifying the system) to the initialization problem.

7 Conclusions and Future Work

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. Third, the effects of retiming and resynthesis on initialization se-

quences were studied. A lag-independent bound was shown on the length increase of initialization sequences of FSMs under retiming; in contrast, unboundability was shown for the case of iterative retiming and resynthesis. In addition, an exact analysis on the length increase was presented.

For future work, it is important to investigate more efficient computation, with reasonable accuracy, of the length increase of initialization sequences for FSMs transformed under retiming and resynthesis. Moreover, as the result of [3] can be modified to obtain a retime function targeting area optimization with minimum increase of initialization sequences, it would be useful to study retiming under other objectives while avoiding increasing initialization sequences.

Acknowledgements

The author wishes to thank Prof. Robert K. Brayton for helpful suggestions.

References

1. G. De Micheli. Synchronous logic synthesis: algorithms for cycle-time minimization. *IEEE Trans. on Computer-Aided Design*, vol. 10, pages 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, pages 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, pages 348–357, March 1996.
4. N. Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*. vol. 17, pages 935–938, 1988.
5. N. Jones. Space-bounded reducibility among combinatorial problems. *Journal of Computer and System Sciences*, vol. 11, pages 68–85, 1975.
6. Z. Kohavi, *Switching and Finite Automata Theory*. McGraw-Hill, 1978.
7. C. E. Leiserson and J. B. Saxe. Optimizing synchronous systems. *Journal of VLSI and Computer Systems*, 1(1):41–67, Spring 1983.
8. C. E. Leiserson and J. B. Saxe. Retiming synchronous circuitry. *Algorithmica*, vol. 6, pages 5–35, 1991.
9. S. Malik. *Combinational Logic Optimization Techniques in Sequential Logic Synthesis*. PhD thesis, University of California, Berkeley, 1990.
10. 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*, vol. 10, pages 74–84, Jan. 1991.
11. M. N. Mneimneh, K. A. Sakallah, and J. Moondanos. Preserving synchronizing sequences of sequential circuits after retiming. In *Proc. ASP-DAC*, Jan. 2004.
12. C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
13. C. Pixley. A theory and implementation of sequential hardware equivalence. *IEEE Trans. Computer-Aided Design*, vol. 11, pages 1469–1478, Dec. 1992.
14. R. K. Ranjan. *Design and Implementation Verification of Finite State Systems*. Ph.D. thesis, University of California at Berkeley, 1997.
15. 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*, pages 402–407, Nov. 1998.

16. W. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, vol. 4, pages 177–192, 1970.
17. 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*, pages 618–625, 1996.
18. V. Singhal, C. Pixley, R. L. Rudell, and R. K. Brayton. The validity of retiming sequential circuits. In *Proc. Design Automation Conference*, pages 316–321, 1995.
19. H. J. Touati and R. K. Brayton. Computing the initial states of retimed circuits. *IEEE Trans. on Computer-Aided Design*, vol. 12, pages 157–162, Jan. 1993.
20. H. Zhou, V. Singhal, and A. Aziz. How powerful is retiming? In *Proc. IWLS*, 1998.