ISSN 1346-5597

NII Technical Report

Synthesis of Timed Circuits based on Decomposition

Tomohiro Yoneda and Chris Myers

NII-2006-001E
Feb. 2006

Synthesis of Timed Circuits based on Decomposition

Tomohiro Yoneda, *National Institute of Informatics, Japan*
Chris Myers, *University of Utah, USA*

***Abstract*— This report presents a decomposition based method for timed circuit design that is capable of significantly reducing the cost of synthesis. In particular, this method synthesizes each output individually. It begins by contracting the timed STG to include only transitions on the output of interest and its possible trigger signals. Next, the reachable state space for this contracted STG is analyzed to determine a minimal number of additional signals which must be reintroduced into the STG to obtain CSC. The circuit for this output is then synthesized from this STG. Results show that the quality of the circuit implementation is nearly as good as the one found from the full reachable state space, but it can be applied to find circuits for which full state space methods cannot be successfully applied. The proposed method has been implemented as a part of our tool `nutas` (Nii-Utah Timed Asynchronous circuit Synthesis system), and its very first version is available at `http://research.nii.ac.jp/~yoneda`.**

***Index Terms*— Decomposition, abstraction, synthesis, timed circuits, timed STGs.**

I. INTRODUCTION

Logic synthesis [1]–[3] from low level specification languages is one of the major approaches to the automated synthesis of asynchronous circuits. This approach can potentially synthesize more optimized circuits with higher performance than other methods such as syntax directed translation methods [4]–[9]. However, it usually requires enumeration of the state space of the given specification, and it often suffers from the state explosion problem. Thus, large specifications expressed in hardware description languages have usually been synthesized by syntax directed translation methods or similar techniques that do not require state space enumeration, sometimes with local optimization techniques such as [10]. This report tackles the challenge of using logic synthesis also for large specifications derived from hardware description languages, as it has the potential of providing further global optimization through timed circuit synthesis [11]. In this approach, a specification written in some high-level language is first translated to a timed signal transition graph (STG), and then, logic synthesis is applied to this timed STG. This method uses a compiler that generates timed STGs with the complete state coding (CSC) property. Its preliminary tool is described in [12], and improved version is described in [13], [14]. Guaranteeing CSC by such a correct-by-construction method, which may not give optimal solutions in the number of inserted state variables, is practical for large STGs, because automatic CSC solvers sometimes do not handle such large STGs well. Furthermore, by using a special protocol shown in [14], [15], the performance degradation caused by the inserted state variables can be reduced to almost negligible amount. A key issue to success of our approach is a new logic synthesis technique that is efficient enough to handle large STGs. This report aims at reducing the average cost for logic synthesis from (timed) STGs by decomposing (or projecting) a specification to many small sub-specifications and running the logic synthesis procedure for each of them.

The idea for decomposition based synthesis is first proposed by Chu [16]. In his work, one primary output is picked up, and the given STG is modified by replacing each transition for the signal that does not affect the output by a dummy transition. Then, the modified STG is reduced by eliminating selected dummy transitions while preserving the behavior. A correct circuit can be synthesized from this reduced STG with usually much smaller cost. He, however, left two open problems. First, the reduction of STGs, called *contraction*, was not formalized. For a simple STG such as a marked graph, its contraction is straightforward. But, in the general case, the formalized algorithm was unknown at that time. Second, it was not straightforward to decide if a signal actually affects the output signal or not, and no algorithm to make this decision is given in his thesis. As for the first problem, Vogler and Wollowski recently formalized the contraction algorithm using a bisimulation relation in [17], and Zheng and Myers developed a timed contraction algorithm in [18]. On the other hand, Puri and Gu tried to solve the second problem in [19]. Their algorithm greedily removes an irrelevant signal (with respect to the output signal) such that the number of CSC violations does not increase by hiding that signal. This algorithm is, however, not so helpful for our purpose, because it needs the state graph of the original STG, which cannot be constructed due to state explosion for very large STGs. Beister, Eckstein, and Wollowski proposed a similar decomposition based method for extended-burst-mode machines [20]. Recently, two separate works, one by Carmona and Cortadella [21] and the other by Khomenjko, Koutny and Yakovlev [22], have been proposed for synthesizing speed-independent circuits efficiently based on an idea similar to that in [19]. Both works first find the necessary input signals (called *support*) for each output by analyzing the original STG, and then synthesize each sub-circuit with each output individually. Unlike the approach of [19], these works do not use the state graph of the original STG explicitly. That is, the original STG is analyzed using the Integer Linear Programming (ILP) technique in the former and

The work of T. Yoneda was supported by JSPS Joint Research Projects. The work of C. Myers was supported by NSF Japan Program award INT-0087281, SRC grant 2002-TJ-1024, and SRC grant 2005-TJ-1357.

the Incremental Boolean Satisfiability (SAT) technique in the latter, and hence, their methods are much more efficient than that of [19].

The main contribution of our work is to propose a new algorithm to find a sufficient set of input signals for a given output for the decomposition based synthesis approach without using the state graph of the original STG. The algorithm starts with a small set of signals which are certainly needed for the output signal, and uses only the state graphs of the contracted STGs for determining other necessary input signals. Since the state graphs of the contracted STGs are usually very small, it does not suffer from the state explosion problem. Furthermore, its decision procedure computes candidates of the necessary signals in many cases more directly than the greedy algorithm in [19], although some cases need heuristics. Since our approach analyzes the state graphs of the contracted STGs explicitly, it is very easy to handle timed STGs, and this is the biggest difference between ours and the above ILP/SAT based approaches. This report describes the theory and the algorithms extended from [23] for the timed circuit synthesis.

The rest of this report is organized as follows. Section II gives several definitions needed for this report, and Section III shows the basic theory of our decomposition based synthesis. Section IV describes the overview of the proposed method, and Section V mentions how to explore timed state spaces and to check synthesizability. Section VI explains in detail how the input sets are determined, which is the main issue of this report. Section VII describes the limitations of our method. Several experimental results are shown in Section VIII, and Section IX gives our conclusion.

II. Synthesizable STGs

A timed STG $G = (P, T, F, \mathsf{Eft}, \mathsf{Lft}, \mu^0, l, \mathit{In}, \mathit{Out})$ is a labeled net, where P is a finite set of *places*, T is a finite set of *transitions* ($P \cap T = \emptyset$), $F \subseteq (P \times T) \cup (T \times P)$ is the *flow relation*, $\mathsf{Eft} : T \to \mathbf{Q}^+$, $\mathsf{Lft} : T \to \mathbf{Q}^+ \cup \{\infty\}$ are functions for the *earliest* and *latest firing times* of transitions satisfying $\mathsf{Eft}(t) \leq \mathsf{Lft}(t)$ for all $t \in T$ ($\mathbf{Q}^+$ denotes the set of nonnegative rationals), $\mu^0 \subseteq P$ is the *initial marking*, $l : T \to (\mathit{In} \cup \mathit{Out}) \times \{+, -\} \cup \{\lambda\}$ is the labeling function, and *In* and *Out* are the input and output signal sets. Let $\mathsf{sig}(G)$ denote $\mathit{In} \cup \mathit{Out}$. A transition t with $l(t) \in \mathit{In} \times \{+, -\}$ is called an *input transition*, t with $l(t) \in \mathit{Out} \times \{+, -\}$ is called an *output transition*, and t with $l(t) = \lambda$ is called a *dummy transition*. For $w \in \mathsf{sig}(G)$, *w-transition* denotes a transition t with $l(t) = w+$ or $w-$. For any transition t, $\bullet t = \{p \in P \mid (p, t) \in F\}$ and $t\bullet = \{p \in P \mid (t, p) \in F\}$ denote the *source places* and the *destination places* of t. For a place p, $\bullet p$ and $p\bullet$ are defined similarly. Transitions t and t' such that $\bullet t \cap \bullet t' \neq \emptyset$ are said to be in *conflict*. Let $\mathit{conflict}(t) = \{t' \mid \bullet t \cap \bullet t' \neq \emptyset\} - \{t\}$. In the rest of this report, when timed STGs G, G_1, etc. are considered, their corresponding components P, T, etc., P_1, T_1, etc. are implicitly considered. Furthermore, a timed STG is simply called an STG, if there is no confusion.

A *marking* μ of G is any subset of P. A transition t is *enabled* in a marking μ if $\bullet t \subseteq \mu$ (all its source places have tokens in μ); otherwise, it is *disabled*. Let $\mathsf{enabled}(\mu)$ be the set of transitions enabled in μ. A *timed state* σ of G is a pair (μ, clock), where μ is a marking and clock is a function $T \to \mathbf{R}^+$ ($\mathbf{R}^+$ denotes the set of nonnegative reals). The *initial timed state* σ^0 is $(\mu^0, \mathsf{clock}^0)$, where $\mathsf{clock}^0(t) = 0$ for all $t \in T$. A timed state changes if time passes or if a transition fires. In timed state $\sigma = (\mu, \mathsf{clock})$, time $\tau \in \mathbf{Q}^+$ can pass, if for all $t \in \mathsf{enabled}(\mu)$, $\mathsf{clock}(t) + \tau \leq \mathsf{Lft}(t)$. In this case, timed state $\sigma' = (\mu', \mathsf{clock}')$ is obtained from σ by passing τ, where

1) $\mu' = \mu$, and
2) for all $t \in T$, $\mathsf{clock}'(t) = \mathsf{clock}(t) + \tau$.

In timed state $\sigma = (\mu, \mathsf{clock})$, transition $t_f \in T$ can fire, if $t_f \in \mathsf{enabled}(\mu)$ and $\mathsf{clock}(t_f) \geq \mathsf{Eft}(t_f)$. In this case, timed state $\sigma' = (\mu', \mathsf{clock}')$ is obtained from σ by firing t_f, where

1) $\mu' = (\mu - \bullet t_f) \cup t_f\bullet$, and
2) for all $t \in T$,
$$\mathsf{clock}'(t) = \begin{cases} 0 & \text{if } t \in \mathsf{enabled}(\mu') - \\ & \qquad \mathsf{enabled}(\mu - \bullet t_f), \\ \mathsf{clock}(t) & \text{else.} \end{cases}$$

That is, firing a transition t_f consumes no time, but updates μ and clock such that the clocks associated with newly enabled transitions (i.e., transitions that are enabled in μ' but not in $\mu - \bullet t_f$) are reset to 0, and clock values of other transitions (i.e., transitions not affected by t_f) are left unchanged. Let $\sigma \xrightarrow{t_f} \sigma'$ denote that σ' is obtained from σ by first passing some time and then firing t_f. For a sequence $v = t_1 t_2 \cdots$ of transitions, $\sigma \xrightarrow{v} \sigma'$ is defined similarly (σ is equal to σ' for an empty v). v is called a *trace*, if there exists a σ' such that $\sigma^0 \xrightarrow{v} \sigma'$. Let $\mathsf{trace}(G)$ denote the set of all traces of G. If there exists a trace that leads to σ', σ' is called *reachable*. A trace may contain multiple occurrences of the same transition. In this report, it is assumed that those occurrences of the same transition are distinguished by some appropriate way, such as, by attaching firing counts, but those are omitted for simplicity in this report. If every reachable timed state $\sigma = (\mu, \mathsf{clock})$ such that there exist t and σ' with $\sigma \xrightarrow{t} \sigma'$ satisfies $(\mu - \bullet t) \cap t\bullet = \emptyset$, then G is called *one-safe*. Intuitively, in a one-safe STG, a token is never produced into a place that is already marked. Furthermore, G is *consistent*, if for every trace $v \in \mathsf{trace}(G)$ and every $w \in \mathsf{sig}(G)$ such that v includes two or more w-transitions, the last two of them are different (i.e., $w+$ and $w-$, or $w-$ and $w+$) [1].

A reachable timed state is mapped to a *signal state*, which is a binary vector representing the values of signals in $\mathit{In} \cup \mathit{Out}$. Different timed states may be mapped to the same signal state. It is sometimes convenient to annotate a signal state with the information whether the outputs are excited to rise or fall. For this purpose, R or F is used in addition to 0 or 1 in signal states. R represents that the corresponding output signal has the binary value of 0, but it is excited to rise. F indicates the signal has a value of 1, but it is excited to fall. When these two notations with or without R/F should be distinguished, we call the former *decorated signal states*, and the latter *nondecorated signal states*. For example, suppose

[1]Remember that $\mathsf{trace}(G)$ is prefi x-closed.

that two timed states σ and σ' have decorated signal states (1010) and $(101R)$ [2]. They have the common nondecorated signal state (1010), but the behavior of the output is different in those timed states. This situation is called a *CSC violation*, and these two timed states are a *CSC violation pair*. If an STG has a CSC violation pair, we say that the STG does not have CSC. Otherwise, it has CSC. If an STG does not have CSC, a circuit cannot be synthesized from the STG without adding a state variable, reducing concurrency, or otherwise changing the behavior of the STG in some way.

This detection of CSC violations is, however, a little complicated, if G has dummy transitions. Suppose σ' is obtained from σ by firing the dummy transition, and that an output signal is excited in σ', but not in σ. σ and σ' have the same nondecorated signal state, while they have different decorated signal states. In this case, however, σ and σ' cannot be distinguished from the outside (i.e., a dummy transition is invisible), and so, it should not be considered that they cause a CSC violation. In order to define this signal excitation formally, it is useful to define a dummy-free version of a state graph. A *timed state graph* of an STG G is a graph $\langle V, E\rangle$ with an initial timed state σ^0, denoted by $\mathcal{G}_G = (\langle V, E\rangle, \sigma^0)$, such that V is the set of all reachable timed states of G, and E is the timed state transition relation of G, that is, $\{(\sigma, t, \sigma') \mid \exists v.\sigma^0 \xrightarrow{v} \sigma, \sigma \xrightarrow{t} \sigma'\}$. A *dummy-free timed state graph* of $\mathcal{G}_G$ is a graph $\langle V', E'\rangle$ with an initial timed state $\sigma^{0'}$, denoted by $\mathcal{G}_G^{df} = (\langle V', E'\rangle, \sigma^{0'})$, satisfying,

1) $\sigma^{0'} = \sigma^0$,
2) $V' = \{\sigma \mid (\sigma', t, \sigma) \in E, t \neq \lambda\} \cup \{\sigma^{0'}\}$,
3) $E' = \{(\sigma, t, \sigma_3) \mid \sigma \in V', (\sigma, u_1u_2\cdots u_n, \sigma_2) \in E^*, n \geq 0, \forall i.u_i = \lambda, (\sigma_2, t, \sigma_3) \in E, t \neq \lambda\}$.

This dummy-free timed state graph is constructed based on the fact that timed state transitions by a (possibly empty) sequence of dummy transitions followed by a nondummy transition can be replaced by the single nondummy transition. Figure 1 (a) shows a simple timed STG G (the transition labeled by t is a dummy transition), and its timed state graph and dummy-free timed state graph are shown in Figures 1 (b) and (c), respectively. Note that $a+$, for example, can fire at any time between 1 and 2 after it becomes enabled, and so, there exist an infinite number of timed states reached from σ_0 by firing $a+$. These figures show for simplicity only timed states that have different markings.

Now, the signal excitation can be defined on this $\mathcal{G}_G^{df} = (\langle V', E'\rangle, \sigma^{0'})$. An output signal w is excited in a timed state σ, if $\exists\sigma'.(\sigma, t, \sigma') \in E'$ with $l(t) = w+$ or $l(t) = w-$. For example, x is excited to rise in σ_1. This straightforward definition is, however, not sufficient for the timed case. Consider σ_2 of Figure 1 (b). In this timed state, both $x+$ and $y+$ are enabled, but, only $x+$ can fire in this state, because the earliest firing time of $y+$ is larger than the latest firing time of $x+$. Thus, σ_1 has only one successor state reached by firing $x+$ in Figure 1 (c). It is, however, necessary to define that y is also excited in σ_1 in order to synthesize a circuit for the output y correctly, because y is triggered by $a+$, not by $x+$, as shown

[2]Note that in (1010), some inputs may be excited, but only outputs are decorated in our definition.

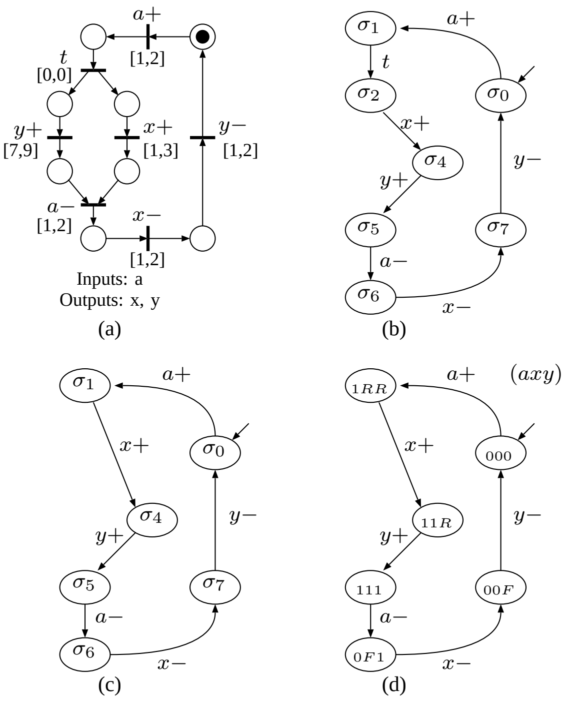
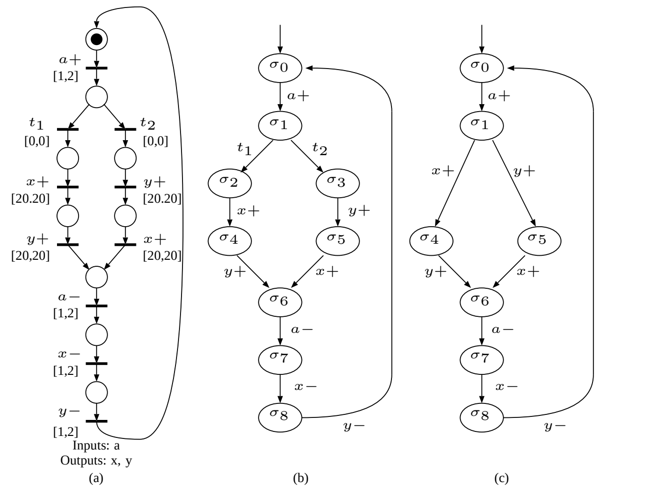


Fig. 1. (a) Simple STG with a dummy transition. (b) Timed state graph. (c) Dummy-free timed state graph. (d) Dummy-free timed state graph with decorated signal states.

in the STG. Therefore, the signal excitation should be defined based on the enabledness information instead of the existence of outgoing edges in the state graph. This is complicated by the fact that either $x+$ or $y+$ is not yet enabled in σ_1.

The definition of signal excitation proposed in this report is as follows. An output signal w is *excited* in a timed state σ of $\mathcal{G}_G^{df} = (\langle V', E'\rangle, \sigma^{0'})$, if there exists a (possibly empty) sequence $u_1u_2\cdots u_n$ of dummy transitions such that $(\sigma, u_1u_2\cdots u_n, \sigma_2) \in E^*$, $\sigma_2 = (\mu_2, \mathsf{clock}_2)$, and $t \in \mathsf{enabled}(\mu_2)$ with $l(t) = w+$ or $l(t) = w-$, where $\mathcal{G}_G = (\langle V, E\rangle, \sigma^0)$. Let $\mathsf{out\_excited}(\sigma)$ be a set of output signals that are excited in σ. Then, it is defined that a timed state σ has R (or F) for an output w in its decorated signal state, if and only if $w \in \mathsf{out\_excited}(\sigma)$ and the binary value of w in σ is 1 (or 0). For example, the decorated signal state of σ_1 in the previous example is $(a, x, y) = (1RR)$. Figure 1 (d) shows decorated signal states on the dummy-free timed state graph. Based on this definition of decorated signal states, the detection of CSC violations can be done in the same way as STGs without dummy transitions.

The property called output semi-modularity is also necessary to synthesize a circuit from an STG. For the untimed case, this property is formally stated as follows. An STG G is output semi-modular, if its dummy-free state graph $\mathcal{G}_G^{df} = (\langle V', E'\rangle, \sigma^{0'})$ satisfies that for any $(\sigma, t, \sigma') \in E'$ with $l(t) = x+$ or $l(t) = x-$, if a signal w $(\neq x)$ is excited in σ, but not in σ', then signals w and x are both input. This definition again has a problem in the timed case. Consider the STG shown in Figure 2 (a), where t_1 and t_2 are dummy transitions. It has the timed state graph and dummy-free timed state graph as shown in Figures 2 (b) and (c).



$a+$ [1,2]
t_1 [0,0]
t_2 [0,0]
$x+$ [20,20]
$y+$ [20,20]
$y+$ [20,20]
$x+$ [20,20]
$a-$ [1,2]
$x-$ [1,2]
$y-$ [1,2]
Inputs: a
Outputs: x, y
(a)
σ_0 $a+$ σ_1 t_1 t_2 σ_2 σ_3 $x+$ $y+$ σ_4 σ_5 $y+$ $x+$ σ_6 $a-$ σ_7 $x-$ σ_8 $y-$
(b)
σ_0 $a+$ σ_1 $x+$ $y+$ σ_4 σ_5 $y+$ $x+$ σ_6 $a-$ σ_7 $x-$ σ_8 $y-$
(c)

Fig. 2. (a) STG with conflicting dummy transitions. (b) Timed state graph. (c) Dummy-free timed state graph.

According to the excitation defined above, in the dummy-free timed state graph, x is excited in both σ_1 and σ_5 as well as y is excited in both σ_1 and σ_4. Thus, this STG satisfies the above property. Hence, if only the untimed behavior of this STG is considered, a circuit such that $a+$ triggers both $x+$ and $y+$ is synthesized from it. However, it is impossible to find any delay assignment to this circuit under which the circuit satisfies the timed behavior of the STG, because when $x+$ fires later than $y+$, $x+$ should fire totally 40 time units later than the firing of $a+$, while otherwise it should fire only 20 time units later than the firing of $a+$. Hence, this timed STG should not be considered to be synthesizable. In this report, we use the following simplified definition of output semi-modularity for timed STGs. A timed STG G is *output semi-modular*, if for any conflicting transitions that are enabled in the same timed state of $\mathcal{G}_G$, every (possibly empty) path of dummy transitions starting from each of them on the STG ends with an input transition. For the STG shown in Figure 2 (a), t_1 and t_2 that are in conflict with each other are enabled in σ_1, and the path of dummy transitions from t_1 on the STG, which is t_1 itself, ends with an output transition $x+$. Thus, this STG is not output semi-modular in our definition. Although this definition is sometimes unnecessarily too strong (e.g., the case that both $x+$ and $y+$ have [0,0] delays in the above STG.), it is, in our experience, not a practical problem.

Although one-safeness of STGs is not required for synthesis, our timed state space enumeration algorithm supports only one-safe STGs like other tools such as petrify [1] and atacs [24]. Furthermore, the consistency significantly simplifies the analysis and synthesis algorithms. Thus, we say that an STG G is *synthesizable*, if G is one-safe, consistent, output semi-modular, and has CSC.

There is another property needed especially for timed circuit synthesis. The timed circuit synthesis method assumes that a synthesized logic function for an output is implemented with a delay within the firing time bounds (i.e., $[\mathsf{Eft}(t), \mathsf{Lft}(t)]$) of the corresponding output transitions in the given timed STG. This assumption, however, may not work, if the output transitions related to the same output signal have different firing time bounds, or even a dummy transition that precedes those output transitions has a non-zero delay. In order to simplify the problem, this report considers a class of timed STGs satisfying the following timed-implementability. A timed STG G is *timed implementable*, if for every output signal x of G, every x-transition has the same firing time bounds, and in any path of dummy transitions on G that ends with an output transition, all dummy transitions have [0,0] bounds.

III. Decomposition theory

There are several definitions of correctness in the context of the decomposition based synthesis. In [16], it is defined that the parallel composition of the contracted STGs for all outputs has the state graph isomorphic to that of the original STG. In [17], it is similar to the conformance used in [25]. Although this correctness is less strict than the Chu's one, STGs must be deterministic (e.g., dummy-free). Our correctness is similar to the latter in the case that the given STG is untimed and deterministic. Ours is, however, different from these two, because the parallel composition is not used for the correctness definition, and STGs with dummy transitions can be naturally handled by defining correctness based on sets of signal states that are obtained from dummy-free timed state graphs. This section gives the formal definition of our correctness and several lemmas and a theorem that are important for our method. It is assumed that the given STG G is synthesizable and timed implementable.

A signal w is a *possible trigger signal* for an output x, if one of its corresponding transitions can reach on G some of x-transitions either directly or through only dummy transitions (i.e., without passing any other signal transitions). Let $\mathsf{trigger}(x)$ denote the set of all possible trigger signals for x.

For $x \in \mathit{Out}$, $\mathit{ES}(x+)$ denotes a set of nondecorated signal states mapped from reachable timed states of a dummy-free timed state graph $\mathcal{G}_G^{df}$ where their decorated signal states have R for x, and $\mathit{QS}(x+)$ denotes a set of similar nondecorated signal states except that their decorated signal states have 1 for x. $\mathit{ES}(x-)$ and $\mathit{QS}(x-)$ are defined similarly. The other nondecorated signal states are unreachable, and this set is denoted by UR. From the definition of CSC, if and only if an STG has CSC, its $\mathit{ES}(x+)$, $\mathit{QS}(x+)$, $\mathit{ES}(x-)$, and $\mathit{QS}(x-)$ are disjoint for each $x \in \mathit{Out}$.

A circuit is defined by a set of logic functions (i.e., the technology mapping is beyond the scope of this report), and a logic function is specified by a *cover*, which is a set of nondecorated signal states where the logic function takes the value 1. In this report, the implementation technologies considered are *atomic gates* and *generalized-C (gC) elements*. In the atomic gate implementation, an STG G defines for each $x \in \mathit{Out}$ a cover, denoted by $C(x)$, satisfying

$$C(x) - \mathit{UR} = \mathit{ES}(x+) \cup \mathit{QS}(x+).$$

In the gC implementation, G defines two covers $C(x+)$ and $C(x-)$ satisfying

$$\mathit{ES}(x+) \subseteq C(x+) - \mathit{UR} \subseteq \mathit{ES}(x+) \cup \mathit{QS}(x+),$$
$$\mathit{ES}(x-) \subseteq C(x-) - \mathit{UR} \subseteq \mathit{ES}(x-) \cup \mathit{QS}(x-).$$

An STG G_1 is *cover-correct* with respect to G, if for each output signal of G_1, the covers $C_1(x)$ or $C_1(x+)$, $C_1(x-)$ for G_1 satisfy the above conditions of the covers for G. For example, in the case of the atomic gate implementation, $C_1(x)$ satisfying $C_1(x) - \mathit{UR}_1 = \mathit{ES}_1(x+) \cup \mathit{QS}_1(x+)$ must satisfy $C_1(x) - \mathit{UR} = \mathit{ES}(x+) \cup \mathit{QS}(x+)$.

In order that a correct delay can be assigned to the synthesized circuit, another property is needed for the correctness of G_1. An STG G_1 is *delay-correct* with respect to G, if G_1 is timed implementable, and for every output signal x of G_1, every x-transition of G_1 has the same firing time bounds (i.e., $[\mathsf{Eft}(t), \mathsf{Lft}(t)]$) as x-transitions in G.

If G_1 is both cover-correct and delay-correct with respect to G, G_1 is *correct* with respect to G. Intuitively, a circuit synthesized from G_1 behaves as expected in G in a sense that the logic function takes the value one in the states if and only if G expects the output to be excited or stable high, as long as the circuit is in the context that G considers. Note that from the timed implementability of G, the delay-correctness of G_1 is easily achieved by disallowing the contraction for the transitions in $\mathsf{trigger}(x) \cup \{x\}$ for $x \in \mathit{Out}_1$. Thus, the rest of this section focuses on the cover-correctness.

For STGs G_1 and G_2 with $\mathit{Out}_1 = \mathit{Out}_2$ and $\mathit{In}_1 = \mathit{In}_2$, a *simulation* from G_1 to G_2 is a relation S between timed states of $\mathcal{G}^{df}_{G_1} = (\langle V'_1, E'_1\rangle, \sigma^0_1)$ and $\mathcal{G}^{df}_{G_2} = (\langle V'_2, E'_2\rangle, \sigma^0_2)$ satisfying

- $(\sigma^0_1, \sigma^0_2) \in S$,
- for any $(\sigma_1, \sigma_2) \in S$, $\mathsf{out\_excited}(\sigma_1) = \mathsf{out\_excited}(\sigma_2)$ holds, and
- for any $(\sigma_1, \sigma_2) \in S$ and any $(\sigma_1, t_1, \sigma'_1) \in E'_1$, there exists some t_2 and σ'_2 such that $l(t_2) = l(t_1)$, $(\sigma_2, t_2, \sigma'_2) \in E'_2$, and $(\sigma'_1, \sigma'_2) \in S$ hold.

Let $G_1 \rightsquigarrow G_2$ denote that G_1 and G_2 have the same input and output signal sets, and that there exists a simulation from G_1 to G_2.

Lemma 1: For STGs G_1 and G_2, if $G_1 \rightsquigarrow G_2$ and G_2 has CSC, then for $x \in \mathit{Out}_1$, the following hold.

$$\begin{array}{ll}
\mathit{ES}_1(x+) \subseteq \mathit{ES}_2(x+), & \mathit{ES}_2(x+) - \mathit{ES}_1(x+) \subseteq \mathit{UR}_1, \\
\mathit{QS}_1(x+) \subseteq \mathit{QS}_2(x+), & \mathit{QS}_2(x+) - \mathit{QS}_1(x+) \subseteq \mathit{UR}_1, \\
\mathit{ES}_1(x-) \subseteq \mathit{ES}_2(x-), & \mathit{ES}_2(x-) - \mathit{ES}_1(x-) \subseteq \mathit{UR}_1, \\
\mathit{QS}_1(x-) \subseteq \mathit{QS}_2(x-), & \mathit{QS}_2(x-) - \mathit{QS}_1(x-) \subseteq \mathit{UR}_1, \\
\mathit{UR}_1 \supseteq \mathit{UR}_2. &
\end{array}$$

Proof: For $s_1 \in \mathit{ES}_1(x+)$, let σ_1 denote a timed state of $\mathcal{G}^{df}_{G_1}$ from which s_1 is mapped, and suppose that σ_1 is reached from σ^0_1 by a sequence v_1 of transitions on $\mathcal{G}^{df}_{G_1}$. From $G_1 \rightsquigarrow G_2$, a timed state, denoted by σ_2, is reachable on $\mathcal{G}^{df}_{G_2}$ by a sequence v_2 of transitions such that $l(v_1) = l(v_2)$, where $l(t_1 t_2 \cdots) = l(t_1) l(t_2) \cdots$ (note that λ is deleted in this sequence). Thus, σ_2 is mapped to also s_1. Since $\mathsf{out\_excited}(\sigma_1) = \mathsf{out\_excited}(\sigma_2)$ and $s_1 \in \mathit{ES}_1(x+)$ hold, $s_1 \in \mathit{ES}_2(x+)$ also holds. Hence, $\mathit{ES}_1(x+) \subseteq \mathit{ES}_2(x+)$ holds. The other three cases can be proved similarly. Next, suppose that $s_2 \in \mathit{ES}_2(x+) - \mathit{ES}_1(x+)$ holds. Since the value for x is 0 in s_2, s_2 is included in either $\mathit{QS}_1(x-)$ or UR_1. If $s_2 \in \mathit{QS}_1(x-)$ holds, then $s_2 \in \mathit{QS}_2(x-)$ holds from $\mathit{QS}_1(x-) \subseteq \mathit{QS}_2(x-)$. This, however, violates that G_2 has CSC from $s_2 \in \mathit{ES}_2(x+)$. Hence, s_2 must be in UR_1. The remaining three cases can be proved similarly. Finally, since additional reachable signal states in G_2 are in UR_1, $\mathit{UR}_1 \supseteq \mathit{UR}_2$ holds. ■

The key of this proof is that the corresponding timed states in $\mathcal{G}^{df}_{G_1}$ and $\mathcal{G}^{df}_{G_2}$ have the same set of excited signals. For untimed methods, the trace equivalence relation is used to guarantee this (e.g. in [21]). For timed methods, however, such a relation on traces are not helpful, because excited transitions does not necessarily fire as shown in Figure 1, i.e., the traces defined by transition firings do not give sufficient excitation information. Hence, the simulation relation defined above is necessary.

For a nondecorated signal state s and a set D of signals, the *D-closure* of s, denoted by $\mathcal{C}_D(s)$, is a set of all nondecorated signal states, including s, such that their binary vectors are the same if the signals in D are projected out. The *core* of a D-closure is the common binary vector obtained by projecting out the signals in D. For example, for $s = (abcd) = (1101)$ and $D = \{a, b\}$, $\mathcal{C}_D(s) = \{0001, 0101, 1001, 1101\}$ and its core is $(cd) = (01)$. The mappings from D-closure $\mathcal{C}_D(s)$ to its core s' and its inverse are defined by $\mathsf{proj}_D(\mathcal{C}_D(s))$ and $\mathsf{proj}^{-1}_D(s')$. Note that both are the one-to-one mappings. The D-closure and these mappings are extended to sets as follows: $\mathcal{C}_D(S) = \bigcup_{s \in S} \mathcal{C}_D(s)$, $\mathsf{proj}_D(\mathcal{C}_D(S)) = \{\mathsf{proj}_D(\mathcal{C}_D(s)) \mid s \in S\}$, and $\mathsf{proj}^{-1}_D(S') = \bigcup_{s' \in S'} \mathsf{proj}^{-1}_D(s')$.

For an STG G and $x \in \mathit{Out}$, a set D of signals is an *irrelevant input set* for x, if

1) $D \subseteq \mathit{In} \cup \mathit{Out} - \{x\}$,
2) $\mathcal{C}_D(\mathit{ES}(x+)) - \mathit{UR} = \mathit{ES}(x+)$, and
3) $\mathcal{C}_D(\mathit{ES}(x-)) - \mathit{UR} = \mathit{ES}(x-)$.

From this definition, the following lemma holds.

Lemma 2: For $x \in \mathit{Out}$ and any irrelevant input set D for x, the following hold.

$$\begin{array}{l}
\mathcal{C}_D(\mathit{QS}(x+)) - \mathit{UR} = \mathit{QS}(x+), \\
\mathcal{C}_D(\mathit{QS}(x-)) - \mathit{UR} = \mathit{QS}(x-).
\end{array}$$

Proof: Suppose $\mathcal{C}_D(\mathit{QS}(x+)) - \mathit{UR} \neq \mathit{QS}(x+)$. From $\mathit{QS}(x+) \subseteq \mathcal{C}_D(\mathit{QS}(x+))$, this means that for some $s \in \mathcal{C}_D(\mathit{QS}(x+)) - \mathit{UR}$, $s \notin \mathit{QS}(x+)$ holds. From $x \notin D$, such s must be in $\mathit{ES}(x-)$. Since $s \in \mathcal{C}_D(\mathit{QS}(x+))$ holds, there exists $s_2 \in \mathit{QS}(x+)$ such that $s \in \mathcal{C}_D(s_2)$. From $s \in \mathcal{C}_D(s)$, $\mathcal{C}_D(s)$ and $\mathcal{C}_D(s_2)$ have a common element, which implies $\mathcal{C}_D(s) = \mathcal{C}_D(s_2)$ and so $s_2 \in \mathcal{C}_D(s)$. From $s \in \mathit{ES}(x-)$ and $\mathcal{C}_D(\mathit{ES}(x-)) - \mathit{UR} = \mathit{ES}(x-)$, $s_2 \in \mathit{ES}(x-)$ is derived, which however, contradicts that G has CSC and so $\mathit{ES}(x-)$ and $\mathit{QS}(x+)$ are disjoint. The remaining case can be proved similarly. ■

For an STG G, $x \in \mathit{Out}$ and a set D of signals with $x \notin D$, let $G_{D,x}$ denote an STG obtained from G by making transitions related to signals in D dummy, which has the input signal set $\mathsf{sig}(G) - D - \{x\}$ and the output signal set $\{x\}$. Let ES_1, QS_1 and so on be for $G_{D,x}$.

Lemma 3: For $x \in \mathit{Out}$ and any irrelevant input set D for x, the following hold.

$$\begin{array}{l}
\mathsf{proj}^{-1}_D(\mathit{ES}_1(x+)) = \mathcal{C}_D(\mathit{ES}(x+)), \\
\mathsf{proj}^{-1}_D(\mathit{ES}_1(x-)) = \mathcal{C}_D(\mathit{ES}(x-)), \\
\mathsf{proj}^{-1}_D(\mathit{QS}_1(x+)) = \mathcal{C}_D(\mathit{QS}(x+)), \\
\mathsf{proj}^{-1}_D(\mathit{QS}_1(x-)) = \mathcal{C}_D(\mathit{QS}(x-)), \\
\mathsf{proj}^{-1}_D(\mathit{UR}_1) \subseteq \mathit{UR}.
\end{array}$$

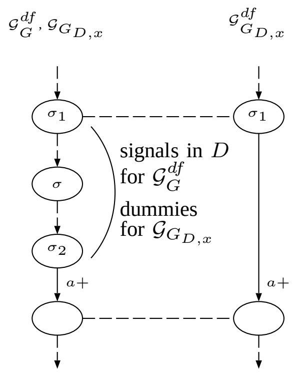

$\mathcal{G}_G^{df}, \mathcal{G}_{G_{D,x}}$
$\mathcal{G}_{G_{D,x}}^{df}$
σ_1
σ_1
σ
σ_2
signals in D for $\mathcal{G}_G^{df}$
dummies for $\mathcal{G}_{G_{D,x}}$
$a+$
$a+$

Fig. 3. Relation between $\mathcal{G}_G^{df}$ and $\mathcal{G}_{G_{D,x}}^{df}$.

Proof: Since $G_{D,x}$ has the same flow relation as G except that some transitions are dummy in $G_{D,x}$, there exists a one-to-one mapping between timed states of $\mathcal{G}_G$ and $\mathcal{G}_{G_{D,x}}$. In the dummy-free timed state graph $\mathcal{G}_{G_{D,x}}^{df}$, however, some timed states that are in $\mathcal{G}_G^{df}$ are deleted from the construction of the dummy-free timed state graphs, as shown in Figure 3. Note that $\mathcal{G}_G^{df}$ and $\mathcal{G}_{G_{D,x}}$ also have a one-to-one mapping if G have no dummy transitions, and Figure 3 shows this case.

The proof of this lemma first shows $\mathit{ES}_1(x+) \subseteq \mathsf{proj}_D(\mathcal{C}_D(\mathit{ES}(x+)))$. Suppose that $s_1 \in \mathit{ES}_1(x+)$, and let σ_1 be the timed state in $\mathcal{G}_{G_{D,x}}^{df}$ that is mapped to s_1. This σ_1 exists also in $\mathcal{G}_G^{df}$ from the above relation between $\mathcal{G}_G^{df}$ and $\mathcal{G}_{G_{D,x}}^{df}$, and let s be its signal state. Then, $s_1 = \mathsf{proj}_D(\mathcal{C}_D(s))$ holds. Furthermore, $s \in \mathit{ES}(x+)$ holds from the following reason. From the signal excitation of $\mathcal{G}_{G_{D,x}}^{df}$, $x+$ must be excited to rise in some timed state reached only by transitions related to the signals in D, such as σ_2 shown in Figure 3. Let s_2 be its signal state. Then, $s_2 \in \mathit{ES}(x+)$ and $s \in \mathcal{C}_D(s_2) - \mathit{UR}$ hold. Since D is an irrelevant input set, $\mathcal{C}_D(\mathit{ES}(x+)) - \mathit{UR} = \mathit{ES}(x+)$ holds, and so, $s \in \mathit{ES}(x+)$ holds. Hence, $s_1 \in \mathsf{proj}_D(\mathcal{C}_D(\mathit{ES}(x+)))$ is derived.

Next, $\mathit{ES}_1(x+) \supseteq \mathsf{proj}_D(\mathcal{C}_D(\mathit{ES}(x+)))$ is proved by showing that for $s \in \mathit{ES}(x+)$, $s_1 = \mathsf{proj}_D(\mathcal{C}_D(s))$ satisfies $s_1 \in \mathit{ES}_1(x+)$. Let σ be the timed state in $\mathcal{G}_G^{df}$ that is mapped to s. This σ may or may not exist in $\mathcal{G}_{G_{D,x}}^{df}$. In the former case, it has the signal state $s_1 = \mathsf{proj}_D(\mathcal{C}_D(s))$ in $\mathcal{G}_{G_{D,x}}^{df}$, and x is excited to rise in σ. Hence, $s_1 \in \mathit{ES}_1(x+)$ is derived. In the latter case, from the construction of dummy-free state graphs, there exists a timed state σ_1 included in both $\mathcal{G}_G^{df}$ and $\mathcal{G}_{G_{D,x}}^{df}$, from which σ is reached only by transitions related to the signals in D (see Figure 3). From this relation between σ and σ_1, σ_1 has the signal state $s' \in \mathcal{C}_D(s) - \mathit{UR}$ and $s_1 = \mathsf{proj}_D(\mathcal{C}_D(s))$ in $\mathcal{G}_{G_{D,x}}^{df}$. Since D is an irrelevant input set, $\mathcal{C}_D(\mathit{ES}(x+)) - \mathit{UR} = \mathit{ES}(x+)$ holds, and so, $s' \in \mathit{ES}(x+)$ holds. Thus, x is excited to rise in σ' in $\mathcal{G}_G^{df}$, and therefore, $s_1 \in \mathit{ES}_1(x+)$ is derived.

Hence, $\mathit{ES}_1(x+) = \mathsf{proj}_D(\mathcal{C}_D(\mathit{ES}(x+)))$ is shown. Applying proj_D^{-1} derives the first property. The proofs for the other three properties are similar.

Furthermore, from the above discussion, if σ mapped to s is reached in $\mathcal{G}_G^{df}$, then some σ' mapped to $s_1 = \mathsf{proj}_D(\mathcal{C}_D(s))$ is also reached in $\mathcal{G}_{G_{D,x}}^{df}$. Thus, if $s_1 \in \mathit{UR}_1$ holds, then every $s \in \mathsf{proj}_D^{-1}(s_1)$ is also in UR. Hence, the final property holds. ■

Lemma 4: For $x \in \mathit{Out}$ and a set D of signals with $D \subseteq \mathit{In} \cup \mathit{Out} - \{x\}$, if $G_{D,x}$ has CSC, and $D \cap \mathsf{trigger}(x) = \emptyset$ holds, then D is an irrelevant input set.

Proof: Suppose that D is not an irrelevant input set. Then, there exist signal states $s \in \mathit{ES}(x+)$ and $s' \in \mathcal{C}_D(s) - \mathit{UR}$ such that $s' \notin \mathit{ES}(x+)$, or there exist signal states $s \in \mathit{ES}(x-)$ and $s' \in \mathcal{C}_D(s) - \mathit{UR}$ such that $s' \notin \mathit{ES}(x-)$. In this proof, the former case is considered, but the latter case can be proved similarly. There exist timed states σ and σ' in $\mathcal{G}_G^{df}$ whose signal states are s and s', respectively. From the construction of the dummy-free timed state graphs, there exist timed state σ_1 and σ_1' in $\mathcal{G}_{G_{D,x}}^{df}$ whose signal states are s_1 and s_1', satisfying $s_1 = \mathsf{proj}_D(\mathcal{C}_D(s))$ and $s_1' = \mathsf{proj}_D(\mathcal{C}_D(s'))$ (see Figure 3). Note that $s_1 = s_1'$ holds from $\mathsf{proj}_D(\mathcal{C}_D(s)) = \mathsf{proj}_D(\mathcal{C}_D(s'))$. These σ_1 and σ_1' cannot be the same from the following reason. σ and σ_1 can be the same, only when one of them is reached from the other on $\mathcal{G}_G^{df}$ only by transitions related to the signals in D. In this case, however, from $s \in \mathit{ES}(x+)$ and $s' \notin \mathit{ES}(x+)$, either $x+$ is disabled without firing $x+$, which violates the output semi-modularity of G, or $x+$ is enabled without firing any transition in $\mathsf{trigger}(x)$ from $D \cap \mathsf{trigger}(x) = \emptyset$, which is impossible. Thus, σ_1 and σ_1' must be different. Since x is excited to rise in σ, so is it in σ_1 of $\mathcal{G}_{G_{D,x}}^{df}$. On the other hand, although x is not excited in σ', x may be considered to be excited to rise in σ_1' of $\mathcal{G}_{G_{D,x}}^{df}$ from the excitation definition, if $x+$ is enabled in some timed state of $\mathcal{G}_{G_{D,x}}$ (such as σ_2 in Figure 3) that is reached from σ' by only dummy transitions. But, this cannot happen, because $x+$ is disabled in σ' and $D \cap \mathsf{trigger}(x) = \emptyset$ holds. Therefore, x is not excited in σ_1' of $\mathcal{G}_{G_{D,x}}^{df}$, either. This contradicts that $G_{D,x}$ has CSC, because x is excited in σ', not excited in σ_1', and $s_1 = s_1'$ holds. Hence, D should be an irrelevant input set. ■

Lemma 5: For $x \in \mathit{Out}$ and any irrelevant input set D for x, suppose that G' satisfies $G_{D,x} \rightsquigarrow G'$. If G' has CSC, then G' is cover-correct with respect to G.

Proof: This proof focuses on the gC implementation, and furthermore, only the cover for $x+$ is considered, because the proofs for the other cases can be done similarly. Let ES_1, QS_1, and UR_1 be for $G_{D,x}$, and ES_2, QS_2, and UR_2 be for G'.

Let $C_2(x+)$ denote the cover for G' and $x+$. From the definition of covers,

$$\mathit{ES}_2(x+) \subseteq C_2(x+) - \mathit{UR}_2 \subseteq \mathit{ES}_2(x+) \cup \mathit{QS}_2(x+). \quad (1)$$

holds. This proof shows that $C_2(x+)$ is also a cover of G. Since $C_2(x+)$ should be considered in the signal state space of G, this is shown by

$$\begin{aligned} \mathit{ES}(x+) &\subseteq \mathsf{proj}_D^{-1}(C_2(x+)) - \mathit{UR} \\ &\subseteq \mathit{ES}(x+) \cup \mathit{QS}(x+). \end{aligned} \quad (2)$$

To show the above, this proof first shows

$$\mathit{ES}_1(x+) \subseteq C_2(x+) - \mathit{UR}_1 \subseteq \mathit{ES}_1(x+) \cup \mathit{QS}_1(x+), \quad (3)$$

and then (2) is shown.

The above (1) is rewritten by removing UR_1 as follows.

$$\begin{aligned} \mathit{ES}_2(x+) - \mathit{UR}_1 &\subseteq C_2(x+) - \mathit{UR}_2 - \mathit{UR}_1 \\ &\subseteq \mathit{ES}_2(x+) \cup \mathit{QS}_2(x+) - \mathit{UR}_1. \end{aligned} \quad (4)$$

From $UR_1 \supseteq UR_2$ as shown in Lemma 1,

$$C_2(x+) - UR_2 - UR_1 = C_2(x+) - UR_1$$

holds. Furthermore, $ES_2(x+) - UR_1$ is rewritten to $ES_1(x+)$ as follows using Lemma 1.

$$\begin{aligned} & ES_2(x+) - UR_1 \\ &= ES_1(x+) \cup (ES_2(x+) - ES_1(x+)) - UR_1 \\ &= ES_1(x+) - UR_1 \\ &= ES_1(x+). \end{aligned}$$

Similarly, $QS_2(x+) - UR_1 = QS_1(x+)$ holds. Hence, (4) is rewritten to (3).

Next, applying proj_D^{-1} to (3) derives

$$\begin{aligned} \mathsf{proj}_D^{-1}(ES_1(x+)) &\subseteq \mathsf{proj}_D^{-1}(C_2(x+) - UR_1) \\ &\subseteq \mathsf{proj}_D^{-1}(ES_1(x+) \cup QS_1(x+)). \end{aligned}$$

From Lemma 3, this is rewritten as

$$\begin{aligned} \mathcal{C}_D(ES(x+)) &\subseteq \mathsf{proj}_D^{-1}(C_2(x+) - UR_1) \\ &\subseteq \mathcal{C}_D(ES(x+)) \cup \mathcal{C}_D(QS(x+)), \end{aligned}$$

and, by removing UR,

$$\begin{aligned} \mathcal{C}_D(ES(x+)) - UR &\subseteq \mathsf{proj}_D^{-1}(C_2(x+) - UR_1) - UR \\ &\subseteq (\mathcal{C}_D(ES(x+)) - UR) \cup (\mathcal{C}_D(QS(x+)) - UR)) \end{aligned}$$

is obtained. Since D is an irrelevant input set, $\mathcal{C}_D(ES(x+)) - UR = ES(x+)$ holds, and from Lemma 2, $\mathcal{C}_D(QS(x+)) - UR = QS(x+)$ holds. Thus,

$$\begin{aligned} ES(x+) &\subseteq \mathsf{proj}_D^{-1}(C_2(x+) - UR_1) - UR \\ &\subseteq ES(x+) \cup QS(x+) \end{aligned} \quad (5)$$

holds. The above $\mathsf{proj}_D^{-1}(C_2(x+) - UR_1) - UR$ can be rewritten as follows from $\mathsf{proj}_D^{-1}(UR_1) \subseteq UR$ by Lemma 3.

$$\begin{aligned} & \mathsf{proj}_D^{-1}(C_2(x+) - UR_1) - UR \\ &= \mathsf{proj}_D^{-1}(C_2(x+)) - \mathsf{proj}_D^{-1}(UR_1) - UR \\ &= \mathsf{proj}_D^{-1}(C_2(x+)) - UR. \end{aligned} \quad (6)$$

Hence, from (5) and (6), (2) is obtained. ■

For an STG G, $x \in Out$, and $V \subseteq \mathsf{sig}(G)$ such that $x \in V$, let $\mathsf{abs}(G, V, x)$ be any STG such that $G_{D,x} \rightsquigarrow \mathsf{abs}(G, V, x)$ with $D = \mathsf{sig}(G) - V$. The main theorem is as follows.

Theorem 1: If $\mathsf{abs}(G, V, x)$ has CSC for V with $\mathsf{trigger}(x) \subseteq V$, then $\mathsf{abs}(G, V, x)$ is cover-correct with respect to G.

Proof: From $\mathsf{trigger}(x) \subseteq V$, $D = \mathsf{sig}(G) - V$ satisfies $D \cap \mathsf{trigger}(x) = \emptyset$. $\mathsf{abs}(G, V, x)$ has CSC. Thus, from Lemma 4, D is an irrelevant input set for x. From the above definition, $G_{D,x} \rightsquigarrow \mathsf{abs}(G, V, x)$ holds, and $\mathsf{abs}(G, V, x)$ has CSC. Hence, from Lemma 5, $\mathsf{abs}(G, V, x)$ is cover-correct with respect to G. ■

IV. Decomposition based synthesis overview

The top level algorithm for the proposed decomposition based synthesis is shown in Figure 4. For a given synthesizable and timed implementable STG G, our algorithm tries to compute an abstraction G_{abs} for each output signal x of G, such that a correct circuit for x can be synthesized from it. Then, a timed circuit synthesis algorithm is applied to G_{abs}.

```
decomposition_based_synthesis($G$) {
  forall $x \in Out$ {
    $G_{abs}$ = obtain_abs($G$, $x$);
    $C_x$ = timed_logic_synthesis($G_{abs}$);
  }
}
```

Fig. 4. Top-level algorithm for synthesis.

```
obtain_abs($G$, $x$) {
  $V$ = $\{x\} \cup \mathsf{trigger}(x)$;
  while(true) {
    $G_{abs}$ = contract_STG($G$, $V$);
    ($res$, $ssg$) = check_synthesizable($G_{abs}$);
    if ($res$ == "synthesizable") return $G_{abs}$;
    if ($res$ == "one-safeness violation") {
      if (no transition can be contracted) abort;
      disallow contraction of some transitions;
      continue;
    }
    if ($res$ == "consistency or O.S.M violation") abort;
    $CSCV$ = obtain_CSC_violation_trace_set($G_{abs}$, $ssg$);
    forall $g \in CSCV$ {
      $candidate$ =
          analyze_CSCV_trace($g$, $G$, $V$, $x$);
      if ($candidate$ == "not found") abort;
      add_constraints_matrix($candidate$);
    }
    $newV$ = solve_covering_problem();
    $V = V \cup newV$;
  }
}
```

Fig. 5. Algorithm to obtain an abstraction.

The algorithm for obtaining such an abstraction is shown in Figure 5. It first constructs the initial input set V for x by taking x and its possible trigger signals. As shown in Theorem 1, the input signal set needs to include those trigger signals in order to obtain a correct abstraction.

The algorithm next contracts dummy transitions in G', if possible, where G' is an STG obtained from G by replacing transitions related to signals in $\mathsf{sig}(G) - V$ by dummy transitions. This contraction of transitions should produce a reduced STG G'' such that $G' \rightsquigarrow G''$ and G'' is delay-correct with respect to G'. Such contraction of timed STGs can be done in a way similar to that shown in [18]. Our method, however, applies the contraction to only transitions with a restricted class of the flow relation, such that consistency and output semi-modularity are preserved. We prefer this *exact contraction* [3], because it synthesizes more optimal circuits than general contraction. The further details about the contraction algorithm are omitted in this report. Note that the resultant STG may contain dummy transitions due to the exact contraction.

The reduced STG G_{abs} obtained by the contraction is then checked if it is synthesizable or not. This process needs to enu-

[3]In order to preserve consistency, it is necessary to keep the timing information exactly. That's why it is called exact contraction. Note that it (or any contraction algorithm of timed STGs) cannot preserve one-safeness in general.

merate its timed state space and construct the decorated signal state graph ssg that corresponds to its dummy-free timed state graph. If G_{abs} is synthesizable, the algorithm returns it, because it is correct with respect to the original STG from Theorem 1. If it has one-safeness violation, some transitions that may cause the one-safeness violation are flaged to show that they should not be contracted. Selecting those transitions depends on the contraction algorithm, but it is not difficult. In the case that every transition is already flaged, the original STG is not one-safe. Thus, the algorithm aborts. Otherwise, the contraction process is restarted. If consistency or output semi-modularity is violated (indicated by "consistency or O.S.M violation"), it is violated also in the original STG, because our contraction is exact. Thus, the algorithm aborts.

The remaining case is that G_{abs} does not simply have CSC. This happens when the input signal set does not contain some relevant signals. In this case, some set of traces of G_{abs} that cause CSC violations, $CSCV$, is extracted [4] from ssg. The algorithm then analyzes each $g \in CSCV$ and tries to find candidate inputs to be added in order to resolve the CSC violation. It fails to find the candidate inputs (indicated by "not found"), when the original STG does not have CSC. In this case, the algorithm aborts. Otherwise, the set $candidate$ contains a set of requirements such that each requirement, which is a set of signals, is satisfied if at least one of the signals in the requirement is added to V. In order to resolve the CSC violation, every requirement must be satisfied. Those requirements are added in the constraint matrix to set up a covering problem. This process is repeated for every CSC violation trace in $CSCV$. Finally, the covering problem is solved for those requirements, and the optimal set of signals are added to V. This V is used to compute a new G_{abs}, and the algorithm repeats the above process.

V. Checking synthesizability

The reduced STG G_{abs} is checked if it is synthesizable or not in **check_synthesizable** as shown in Figure 5. This is done by exploring the timed state space of G_{abs} and obtaining its corresponding decorated signal state graph. Since the timed state space of a timed STG is potentially infinite, equivalence classes of timed states are actually explored. Let I be a set of inequalities of the form $t - u \leq c$, where t and u are variables to represent the next firing times of transitions t and u, and c is a constant. For a given marking μ, if I over the variables related to the transitions enabled in μ is considered, then I determines the bounds of the firing time separation of those transitions. Thus, (μ, I) represents an equivalence class of timed states, which is called a *timed state class*. Let $\alpha_0 = (\mu_0, I_0)$ be the initial timed state class, and for a timed state class α, $\mathsf{firable}(\alpha)$ denotes the set of *firable* transitions, i.e., the set of transitions that can fire earlier than any other transition in α. It is known that the timed state class space is finite [26], [27], and so, its space enumeration is done by firing every firable transition from α_0 until no new timed state classes are reached. The inclusion of timed state classes is considered in this process. A timed state class (μ, I) *includes* another timed state class (μ', I'), if $\mu = \mu'$ and the solution set of I includes that of I'. If a timed class α that is newly generated is included by some timed class α' that is generated previously, the traversal from α is stopped. On the other hand, if α includes α', then α' is removed, the arcs from the predecessors of α' to α' are reconnected to α, and the traversal from α is continued.

When enumerating the equivalence classes, one-safeness can be easily checked. Once the graph of this equivalence classes is constructed, its dummy free version is obtained by the way explained in Figure 1. This modified graph is then projected to a decorated signal state graph by considering only decorated signal states. It is straightforward to check the consistency, output semi-modularity, and CSC on this decorated signal state graph.

The above timed state class enumeration can be improved using the ideas of the partial order reduction and POSET method, which are similar to that proposed in [28] and [29]. Since the firing order of dummy transitions does not affect the dummy-free timed state class graph if they are concurrent with any other transitions, the state space explored can be reduced by only considering a single interleaving of firable those dummy transitions. This is effective especially in our case, because the exact timed contraction can contract only a restricted class of dummy transitions, and many dummy transitions sometimes remain in G_{abs}.

VI. Analyzing CSC violation trace

If G_{abs} does not have CSC, a set of CSC violation traces is constructed by **obtain_CSC_violation_trace_set** from the decorated signal state graph. Each of such CSC violation traces is analyzed by **analyze_CSCV_trace**, which is the core part of this report.

The algorithm **analyze_CSCV_trace** first generates a concrete trace of the original STG G which corresponds to the given CSC violation trace of G_{abs}. This is done by a technique similar to the one developed for the partial order reduction, which we call *guided simulation.* Then, it finds a set of requirements for an appropriate input set by analyzing the concrete CSC violation trace. The overall procedure is shown in Figure 6.

This section first discusses the algorithm to analyze the concrete CSC violation traces, because guided simulation is strongly related to this algorithm. The guided simulation is then discussed. In the following, an *interface signal* means the signals used in G_{abs}, i.e., the signals in V, and a *noninterface signal* means the remaining signals of G, i.e., the signals in $D = \mathsf{sig}(G) - V$. The corresponding transitions are called similarly.

A. Regular concrete traces

Each CSC violation trace g of G_{abs}, constructed by **obtain_CSC_violation_trace_set**, is assumed to be of the form $g = \langle g_0, g_1, g_2 \rangle$, $s_0 \xrightarrow{g_0} s_1$, $s_1 \xrightarrow{g_1} s_2$, and $s_2 \xrightarrow{g_2} s_3$, where s_0 is the initial signal state of G_{abs}, s_1 and s_2 are the first two signal states that correspond to the CSC violation pair, and

[4]In our current implementation, one shortest trace is selected for each CSC violation pair, because using all CSC violation traces is very expensive.

```
analyze_CSCV_trace($g$, $G$, $V$, $x$) {
    $f$ = guided_sim_phase1($g$, $G$, $V$, $x$, $\alpha_0$, $null$);
    $can$ = find_inputs($f$, $G$, $V$, $x$);
    if ($can == false$) return "not found";
    else return $can$;
}
```

Fig. 6. Algorithm for analyzing CSC violation trace.

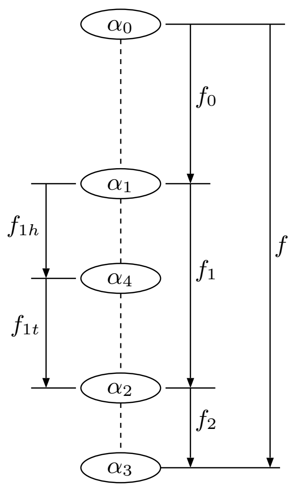


Fig. 7. Labeling of a concrete trace.

g_2 contains exactly one (interface) transition. For this g, the corresponding concrete trace f is constructed by the guided simulation shown later, where $f = \langle f_0, f_1, f_2 \rangle$ such that for $0 \le i \le 2$, projecting out noninterface and dummy transitions from f_i is equal to g_i. It is assumed that f is separated into f_i's when interface transitions fire, i.e., each f_i ends with an interface transition. Let α_1, α_2, and α_3 denote the timed state classes of G obtained by f_0, f_1, and f_2, respectively (see Figure 7). Since some noninterface or dummy transitions fire concurrently with the interface transitions, there exist many such concrete traces that correspond to g. Thus, we first define an equivalence class of traces based on the causality relation.

For two interface transitions a and b in f, if a fires before b without any interface transitions between them, it is denoted by $(a, b) \in R_1^f$. For any two transitions t_1 and t_2 in f, if t_2 fires by consuming the token produced by the firing of t_1, it is denoted by $(t_1, t_2) \in R_2^f$. In the untimed case, the actual causality relation for f is defined by the transitive closure of the union of R_1^f and R_2^f, i.e., $(t_1, t_2) \in (R_1^f \cup R_2^f)^*$. In the timed case, however, it is further needed to consider *timed causality*, which is defined in [30]. For any two transitions t and u with $(t, u) \in (R_2^f)^*$, let $\mathsf{path}_f(t, u)$ denote a set of paths from t to u defined by the relation R_2^f, where each path is represented by a set of transitions. That is,

$$\mathsf{path}_f(t,u) = \begin{cases} \emptyset & \text{if } t = u, \\ \{\{u\} \cup q \mid (t', u) \in R_2^f, & \text{else.} \\ \qquad q \in \mathsf{path}_f(t, t')\} & \end{cases}$$

For example, if f is a concrete trace obtained from the STG shown in Figure 9 (a), then $\mathsf{path}_f(c+, x-)$ is $\{\{a+, x+, a-, x-\}, \{a+, b-, a-, x-\}\}$. Furthermore, let

$$\mathsf{max\_eft}_f(t, u) = max_{q \in \mathsf{path}_f(t,u)} \big(\textstyle\sum_{t \in q} \mathsf{Eft}(t) \big),$$
$$\mathsf{max\_lft}_f(t, u) = max_{q \in \mathsf{path}_f(t,u)} \big(\textstyle\sum_{t \in q} \mathsf{Lft}(t) \big).$$

Finally, for any two transitions t_1 and t_2 in f, if there exists a transition t_3 in f such that $(t_3, t_1) \in (R_2^f)^*$, $(t_3, t_2) \in (R_2^f)^*$, and $\mathsf{max\_lft}_f(t_3, t_1) < \mathsf{max\_eft}_f(t_3, t_2)$, then it is denoted by $(t_1, t_2) \in R_3^f$. Intuitively, $(t_1, t_2) \in R_3^f$ implies that t_1 and t_2 has a common ancestor t_3, and the maximal time separation between t_3 and t_1 is smaller than the minimal time separation between t_3 and t_2. Thus,t_1 cannot fire before t_2 under the causality relation R_2^f. This timed causality relation can be computed using a time separation algorithm such as shown in [11].

Using the above relations, we say that t_1 is an *ancestor* of t_2 in f, denoted by $[t_1 \leadsto_f t_2]$, if t_1 and t_2 are related by the transitive closure of the union of R_1^f, R_2^f and R_3^f. This ancestor relation represents an actual causality relation with respect to the specific abstracted trace g. In the rest of this report, we use the following terminology.

- t_1 *causes* t_2, if $[t_1 \leadsto_f t_2]$ holds.
- t_1 and t_2 are *ordered*, if either $[t_1 \leadsto_f t_2]$ or $[t_2 \leadsto_f t_1]$ holds.
- t_1 and t_2 are *concurrent*, if they are not ordered.

This actual causality relation defines equivalent classes of traces of G. For a trace f, let $\|f\|_G$ denote a set of traces of G (including f) such that for any $f' \in \|f\|_G$, $(R_1^{f'} \cup R_2^{f'} \cup R_3^{f'})^* = (R_1^f \cup R_2^f \cup R_3^f)^*$ holds.

For a given abstracted trace g, our algorithm constructs one particular concrete trace f satisfying a property which we call regularity, and analyzes it to handle all traces in $\|f\|_G$. A trace f is *regular*, if for every interface transition t in f, all noninterface or dummy transitions that are concurrent with t fire before t in f. This regularity is necessary for the following reason. As shown later, for a transition t, which is an ancestor of an interface transition x in f, it is necessary to find a noninterface signal w such that every w-transition in f is ordered with t. If every w-transition appears in f, it is easy to check the above property, i.e., this can actually be done by constructing a data structure similar to an occurrence net [31]. Otherwise, however, it is not clear when the generation of f should be terminated to check the above property with respect to every w-transition, if f is nonregular, because a concurrent w-transition may fire a long time later. On the other hand, if f is regular, every transition concurrent with x is fired before x. Hence, if a regular trace f is generated up to x, it can be decided whether every w-transition is ordered with t or not, because a w-transition that does not appear in f, if it exists, is caused by x, and so, it is caused by t from $[t \leadsto_f x]$.

B. Determining the input set

The idea to resolve the CSC violation between α_1 and α_2 is to add to the input set a noninterface signal w such that f_1 contains odd number of w-transitions. If w-transition fires in f_1 in odd times, then the signal takes different values in α_1 and α_2, and so, the CSC violation is resolved by adding w to the input set. However, such w may not work for other traces

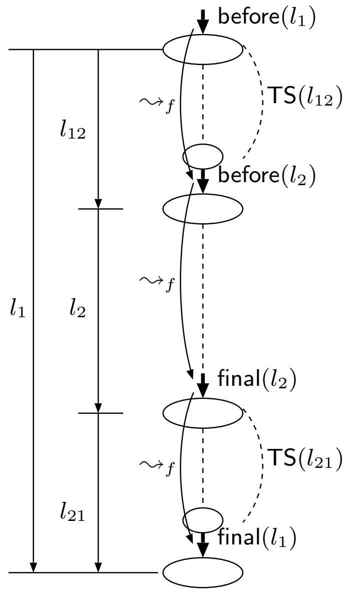


Fig. 8. A nested subtrace pair.

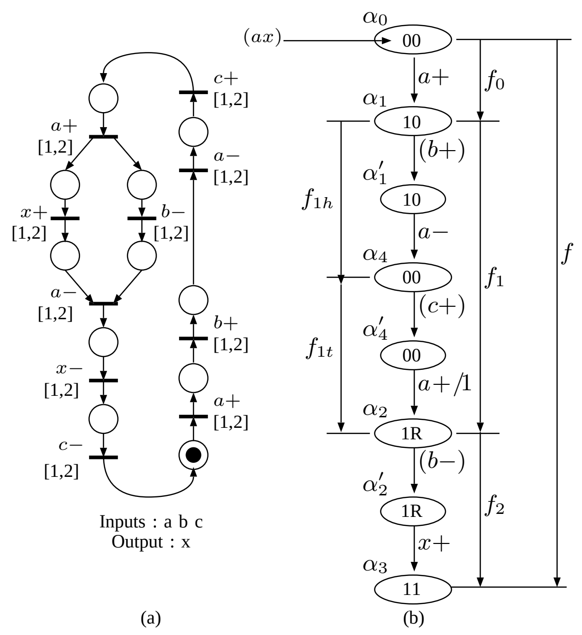


Fig. 9. (a) A simple STG. (b) Its CSC violation trace.

in $\|f\|_G$, unless the causality relation guarantees that it fires certainly odd-times in traces in $\|f\|_G$. Thus, we need to define the following notions.

For a (sub)trace h, let $\mathsf{final}(h)$ denote the last transition in h, and $\mathsf{before}(h)$ the transition fired just before the first transition of h in a currently designated trace. When h starts from the initial timed state class, $\mathsf{before}(h)$ is the virtual transition v that is assumed to cause the first transition of every trace. For a trace f, (l_1, l_2) is a *nested subtrace pair* of f, if $\mathsf{before}(l_1)$, $\mathsf{before}(l_2)$, $\mathsf{final}(l_2)$, and $\mathsf{final}(l_1)$ are distinct, and they are caused in this order, i.e., $[\mathsf{before}(l_1) \rightsquigarrow_f \mathsf{before}(l_2)]$, $[\mathsf{before}(l_2) \rightsquigarrow_f \mathsf{final}(l_2)]$, $[\mathsf{final}(l_2) \rightsquigarrow_f \mathsf{final}(l_1)]$ (see Figure 8). For a signal w and a nested subtrace pair (l_1, l_2) of f, w is *semi-essential* with respect to (l_1, l_2) in f, if

- none of $\mathsf{before}(l_1)$, $\mathsf{before}(l_2)$, $\mathsf{final}(l_2)$, and $\mathsf{final}(l_1)$ is a w-transition, and
- every w-transition is ordered with $\mathsf{before}(l_1)$, $\mathsf{before}(l_2)$, $\mathsf{final}(l_2)$, and $\mathsf{final}(l_1)$.

Similarly, w is *essential* with respect to (l_1, l_2) in f, if w is semi-essential with respect to (l_1, l_2) in f, and l_2 contains an odd number of w-transitions while neither l_{12} nor l_{21} contains any w-transition, where l_{12} and l_{21} are the subtraces of l_1 before l_2 and after l_2 as shown in Figure 8. As mentioned previously, it is easy to check whether w is (semi-)essential or not, if f is regular and contains interface transitions at the end of l_1 or later.

For $f' \in \|f\|_G$, a nested subtrace pair (l_1', l_2') of f' that corresponds to (l_1, l_2) in f is the nested subtrace pair defined by using $\mathsf{before}(l_1)$, $\mathsf{before}(l_2)$, $\mathsf{final}(l_2)$, and $\mathsf{final}(l_1)$. For a subtrace l, let $\mathsf{TS}(l)$ denote a set of timed state classes in which the transitions in l fire (see Figure 8). $\mathsf{SS}(l)$ denotes the corresponding signal state set. The following lemma holds.

Lemma 6: Let (l_1, l_2) be a nested subtrace pair of f and w be essential with respect to (l_1, l_2). For any $f' \in \|f\|_G$ and the nested subtrace pair (l_1', l_2') of f' that corresponds to (l_1, l_2), the signal states in $\mathsf{SS}(l_{12}')$ are distinguished from those in $\mathsf{SS}(l_{21}')$ by the signal w.

Proof: The proof is straightforward from the definition of the essentialness. ■

Consider the first interface transition in f_1, and divide f_1 into f_{1h} and f_{1t} with it, i.e., $f_1 = \langle f_{1h}, f_{1t}\rangle$ and f_{1h} ends with this first interface transition in f_1. Figure 7 shows the relation among $f_0 \cdots f_2$ as well as f_{1h} and f_{1t}. Let $l_1 = \langle f_1, f_2\rangle$ and $l_2 = f_{1t}$. Then, $l_{12} = f_{1h}$ and $l_{21} = f_2$ hold. From the definitions of f_0, f_{1h}, f_1, and f_2, they end with interface transitions. Thus, each of them are different, and they are caused in this order. Hence, $(l_1, l_2) = (\langle f_1, f_2\rangle, f_{1t})$ is a nested subtrace pair of f. The following theorem holds.

Theorem 2: The CSC violation with respect to $f' \in \|f\|_G$ with $f = \langle f_0, f_1, f_2\rangle$ is resolved by adding a noninterface signal w to the input set, if w is essential with respect to $(l_1, l_2) = (\langle f_1, f_2\rangle, f_{1t})$ in f.

Proof: For any $f' \in \|f\|_G$ and the nested subtrace pair (l_1', l_2') of f' that corresponds to (l_1, l_2), the CSC violation is caused between the timed state classes in $\mathsf{TS}(l_{12}')$ and those in $\mathsf{TS}(l_{21}')$. Those signal states are distinguished by w from Lemma 6. ■

For example, consider an STG shown in Figure 9 (a). The output x has the possible trigger signal a, and so, the initial V is $\{a, x\}$. Then, the reduced STG with interface signals a and x has one CSC violation trace, and its corresponding concrete trace f is shown in Figure 9 (b). This trace is regular, because a noninterface transition $b-$ is concurrent with an interface transition $x+$, and $b-$ fires before $x+$ in this trace. $\|f\|_G$ contains another trace f' obtained by swapping $b-$ and $x+$ in f. $(\langle f_1, f_2\rangle, f_{1t})$ is a nested subtrace pair of f, and the noninterface signal c is essential with respect to it. The noninterface signal b is not semi-essential, because it is not ordered with $\mathsf{final}(l_1) = x+$. $\mathsf{TS}(l_{12}) = \{\alpha_1, \alpha_1'\}$ and $\mathsf{TS}(l_{21}) = \{\alpha_2, \alpha_2'\}$ cause the CSC violation, and it is resolved

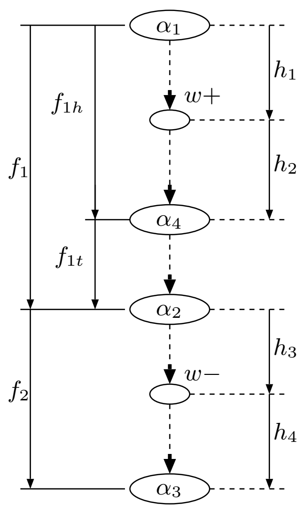


Fig. 10. A nested subtrace pair reduced from the original nested subtrace pair.

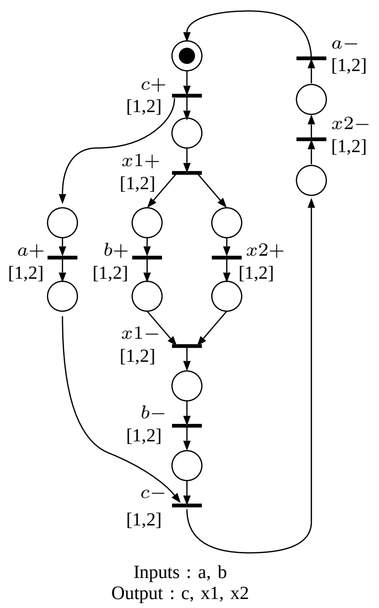


Fig. 11. An STG that has no essential signals.

by adding the essential signal c to V. Actually, for this new $V = \{a, c, x\}$, the reduced STG has CSC, and a circuit for x can be synthesized from it.

If there is no essential signal with respect to $(\langle f_1, f_2\rangle, f_{1t})$ in f, the CSC violation cannot be resolved by adding a single noninterface signal. However, CSC violations can be resolved by adding two or more noninterface signals to the input set. There are two cases depending on the existence of semi-essential noninterface signals.

If there exist in f_{1h} or f_2, noninterface signals that are semi-essential with respect to $(\langle f_1, f_2\rangle, f_{1t})$, the problem can be reduced to several sub-problems that can be solved by the above approach. Suppose that such a semi-essential noninterface signal w changes as shown in Figure 10. Then, by adding w to the input set, the timed state classes that cause CSC violation (i.e., those in $\mathsf{TS}(f_{1h})$ and $\mathsf{TS}(f_2)$) are divided into two groups: one is $\mathsf{TS}(h_1)$ and $\mathsf{TS}(h_4)$, and the other is $\mathsf{TS}(h_2)$ and $\mathsf{TS}(h_3)$. The first group can be handled by considering a nested subtrace pair $(\langle f_1, f_2\rangle, \langle h_2, f_{1t}, h_3\rangle)$. Even if there exists no essential noninterface signal for $(\langle f_1, f_2\rangle, f_{1t})$, this $(\langle f_1, f_2\rangle, \langle h_2, f_{1t}, h_3\rangle)$ may have it, because $\langle h_2, f_{1t}, h_3\rangle$ is longer than f_{1t}. Similarly, the second group can be handled by a nested subtrace pair $(\langle h_2, f_{1t}, h_3\rangle, f_{1t})$, and it may have an essential noninterface signal, because $\langle h_2, f_{1t}, h_3\rangle$ is shorter than $\langle f_1, f_2\rangle$. We say that $(\langle f_1, f_2\rangle, \langle h_2, f_{1t}, h_3\rangle)$ or $(\langle h_2, f_{1t}, h_3\rangle, f_{1t})$ is *reduced* from $(\langle f_1, f_2\rangle, f_{1t})$ with respect to a semi-essential signal w. As mentioned above, it is important that a reduced nested subtrace pair may have an essential noninterface signal, even if the original nested subtrace pair does not. If every reduced nested subtrace pair has an essential noninterface signal, adding those essential signals as well as w to the input set resolves the CSC violation in $\|f\|_G$. If there exists no essential noninterface signal for some reduced nested subtrace pair (l_1, l_2), the above process can be applied to it as long as semi-essential noninterface signals exist for (l_1, l_2), which may solve the CSC violation using more noninterface signals.

Second, if there exists in f_{1h} or f_2 no noninterface signal that is semi-essential with respect to $(\langle f_1, f_2\rangle, f_{1t})$, a more complicated process is necessary to resolve the CSC violation in $\|f\|_G$. For such a nested subtrace pair (l_1, l_2), our algorithm chooses a noninterface transition t fired in f, such that t is concurrent with a transition in a set $\{\mathsf{before}(l_1), \mathsf{before}(l_2), \mathsf{final}(l_2), \mathsf{final}(l_1)\}$. Let u denote such a transition in the above set. It then generates two traces f' and f'' from f by interleaving t and u. The idea is that our algorithm tries to predict from those interleavings the situation where the noninterface signal w related to t is added to the input set and its every transition is ordered with the other interface transitions, before actually doing it. Interleaving t and u can be done as follows. Suppose that t and u fire in this order in f. The new trace f' is obtained simply by adding $[t \rightsquigarrow_f u]$ to the ancestor relation of f. Similarly, f'' is obtained by adding $[u \rightsquigarrow_f t]$ to the ancestor relation of f, but in order to make the firing order of f'' consistent with this modified ancestor relation, it is necessary to move t and the noninterface transitions caused by t and fired before u, next to u.

For example, consider the STG shown in Figure 11 and its output c. The possible trigger signals for c are a and b. For $V = \{a, b, c\}$, G_{abs} has a CSC violation trace $g = c+\ a+\ b+\ b-\ c-$ with $g_0 = c+\ a+$, $g_1 = b+\ b-$ and $g_2 = c-$. The guided simulation generates $f_0 = c+\ x1+\ x2+\ a+$, $f_1 = b+\ x1-\ b-$, and $f_2 = c-$ as shown in Figure 12 (a). Our algorithm first looks for a semi-essential noninterface signal for $(\langle f_1, f_2\rangle, f_{1t})$, but both noninterface transitions $x1+$ and $x2+$ are concurrent with $\mathsf{before}(\langle f_1, f_2\rangle) = a+$. Thus, this STG has no semi-essential signals. Here, choose $x1$ and $\mathsf{before}(\langle f_1, f_2\rangle) = a+$, and generate f' and f'' by interleaving them. It is not easy to illustrate these traces in a figure like Figure 12, because it does not show the ancestor relation precisely, but assume that in Figure 12, $x1+$ causes $a+$ in f' while $a+$ causes $x1+$ in f''. Note that in f'', $x1+$ and $x2+$

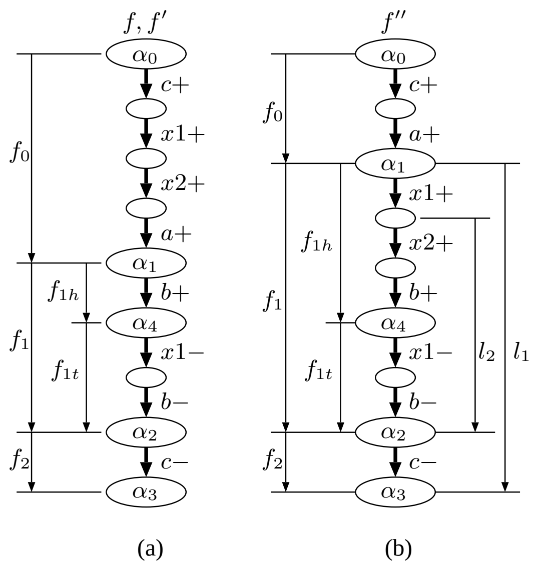


Fig. 12. (a) Original traces. (b) Newly generated traces.

(which is caused by $x1+$ and fired before $a+$) are moved next to $a+$. In f', $x1$ is now essential with respect to $(\langle f_1, f_2\rangle, f_{1t})$, because every $x1$-transition is ordered with $a+$, $b+$, $b-$, $c-$, and only one $x1-$ exists in f_{1t}. Thus, this CSC violation can be resolved in $\|f\|_G$ by adding $x1$ in the input set. In f'', $x1$ is semi-essential, but not essential, because $x1$ fires in f_{1h}. This corresponds to the first case above, and can be handled by considering one reduced nested subtrace pair (l_1, l_2) as shown in Figure 12 (b), because only α_1 and α_2 cause the CSC violation after adding $x1$. Then, our algorithm looks for another noninterface transition that is essential with respect to (l_1, l_2) in f'', which is $x2$ in this case. In other words, $x2$ can resolve the CSC violation between $\mathsf{TS}(l_{12}) = \{\alpha_1\}$ and $\mathsf{TS}(l_{21}) = \{\alpha_2\}$. Therefore, the CSC violation with respect to f'' can be resolved in $\|f''\|_G$ by adding both $x1$ and $x2$.

In general, the above process should be repeated by generating new traces by interleaving some concurrent transitions. Furthermore, there are usually many choices for noninterface signals. Thus, generating as many such combinations as possible is desirable. The whole process for both cases is described by the pseudo code shown in Figure 13. This procedure constructs a Boolean expression E over a set of noninterface signals such that for each feasible assignment of E, the CSC violation is resolved in $\|f\|_G$ by adding the noninterface signals that have 1 in the assignment. Then, it is converted to a conjunctive normal form (CNF). Thus, one noninterface signal in each clause of the CNF should be added to resolve the CSC violation. Hence, this CNF represents a set of requirements, and solving the covering problem that all these requirements are satisfied obtains the optimal set of signals to be added, which is done in **obtain_abs** as mentioned previously.

The following theorem holds.

Theorem 3: The algorithm shown in Figure 13 returns "$false$", only when the given STG have no CSC.

Proof: It can return "$false$", only when there exists

```
find_input($f$, $G$, $V$, $x$) {
   $A$ = build_ancestor_relation($f$, $G$);
   $\langle f_0, f_1, f_2\rangle = f$;
   $\langle f_{1h}, f_{1t}\rangle = f_1$;
   $E$ = find_essential($\langle f_1, f_2\rangle$, $f_{1t}$, $f$, $A$);
   return convert_to_CNF($E$);
}

find_essential($l_1$, $l_2$, $f$, $A$) {
   $E = false$;
   for each noninterface signal $w$ {
      if ($w$ is essential w.r.t. $(l_1, l_2)$ in $f$)
         $E = E \vee w$;
   }
   if ($E \neq false$) return $E$;
   for each noninterface signal $w$ {
      if ($w$ is semi-essential w.r.t. $(l_1, l_2)$ in $f$) {
         $F = w$;
         $N$ = reduce_nested_subtrace_pairs($w$, $l_1$, $l_2$, $f$);
         forall $(h_1, h_2) \in N$ {
            $F'$ = find_essential($h_1$, $h_2$, $f$, $A$);
            $F = F \wedge (F')$;
         }
         $E = E \vee F$;
      }
   }
   if ($E \neq false$) return $E$;
   $(t, u)$ = pick_concurrent_transitions($l_1$, $l_2$);
   if (such $(t, u)$ does not exist) return $false$;
   $N$ = generate_interleavings($t$, $u$, $f$);
   $E = true$;
   forall $(l'_1, l'_2, f', A') \in N$ {
      $F$ = find_essential($l'_1$, $l'_2$, $f'$, $A'$);
      $E = E \wedge (F)$;
   }
   return $E$;
}
```

Fig. 13. Algorithm for determining the input set.

no noninterface transition that is concurrent with some of $\mathsf{before}(l_1)$, $\mathsf{before}(l_2)$, $\mathsf{final}(l_2)$, and $\mathsf{final}(l_1)$. It implies that every noninterface signal except for the signals related to $\mathsf{before}(l_1)$, $\mathsf{before}(l_2)$, $\mathsf{final}(l_2)$, and $\mathsf{final}(l_1)$ is semi-essential. Since no essential signal exists and no reduced nested subtrace pair works, such every noninterface signal included in $\langle f_1, f_2\rangle$ must appear in f_{1t} even times. Hence, the CSC violation in $\mathsf{TS}(f_{1h})$ and $\mathsf{TS}(f_2)$ cannot be resolved, even if every noninterface signal is used. This implies that the given STG has no CSC. ■

C. Guided simulation

For a given abstracted trace g, the guided simulation obtains a regular trace f of G such that a trace obtained by projecting out the noninterface and dummy transitions from f is equal to g. The algorithm is shown in Figure 14. It consists of two phases. The phase 1 generates a concrete trace h that satisfies the projection condition but not the regularity. Actually, for each interface transition t appearing in g, the noninterface and dummy transitions that cause t (by R_2^f and R_3^f) are certainly contained before t in h, but those that are concurrent with t may either appear after t or not appear in h. In the phase 2,

```
**guided_sim**($g$, $G$, $V$, $x$) {
    $h$ = guided_sim_phase1($g$, $G$, $V$, $x$, $\alpha_0$, $null$);
    $f$ = guided_sim_phase2($h$, $G$, $V$, $x$, $\alpha_0$);
    return $f$;
}

**guided_sim_phase1**($g$, $G$, $V$, $x$, $\alpha$, $h$) {
    if ($g$ is empty) return $h$;
    if ($(g, \alpha)$ is already visited) return "backtrack";
    $g_1$ = head($g$);
    $nec$ = necessary($\alpha$, $g_1$);
    $dep$ = $\emptyset$;
    forall $t \in nec$
        $dep$ = $dep \cup$ dependent($\alpha$, $t$);
    forall $t \in dep$ {
        $\alpha'$ = fi re($\alpha$, $t$);
        if ($t$ == $g_1$) $g' = $ tail($g$);
        else $g'$ = $g$;
        $h' = $ append($h$, $t$);
        $result$ = guided_sim_phase1($g'$, $G$, $V$, $x$, $\alpha'$, $h'$);
        if ($result \neq$ "backtrack") return $result$;
    }
    return "backtrack";
}

**dependent**($\alpha$, $t$) {
    $E = new = \{t\}$;
    while(true) {
        $new' = \emptyset$;
        forall $x \in new$
            forall $y \in$ *conflict*$(x) \cap$ *NonIF_Dum*
                $new' = new' \cup$ necessary($\alpha$, $y$) $- E$;
        if ($new' == \emptyset$) break;
        $E = E \cup new'$;
        $new = new'$;
    }
    return $E$;
}

**necessary**($\alpha$, $t$) {        /* $\alpha = (\mu, I)$ */
    if ($t$ is already visited) return $\emptyset$;
    if ($t \in$ enabled($\mu$))
        if ($t \in$ firable($\alpha$)) return $\{t\}$;
        else return {choose_one(firable($\alpha$) $\cap$ *NonIF_Dum*)};
    $E = \emptyset$;
    $p$ =choose_one($\bullet t - \mu$);
    forall $x \in \bullet p \cap$ *NonIF_Dum*
        $E = E \cup$ necessary($\alpha, x$);
    return $E$;
}
```

Fig. 14. Algorithm for guided simulation (1).

the concurrent noninterface or dummy transitions are added to h or moved in order to satisfy the regularity.

In **guided_sim_phase1**, if g is nonempty, it picks the first transition of g, denoted by g_1, and computes its necessary set, i.e., the set of transitions that should be fired for firing g_1, by **necessary**. In **necessary**, if t is not enabled, one of its empty source places is traversed upward along the noninterface or dummy transitions (*NonIF_Dum* denotes the set of all noninterface and dummy transitions) recursively. If t is enabled and firable, it is returned. If t is enabled, but not firable, some firable transition must precede g_1, and so, one of the firable noninterface or dummy transitions chosen by **choose_one** is returned. In the net shown in Figure 15, where $t_1 \cdots t_5$ are

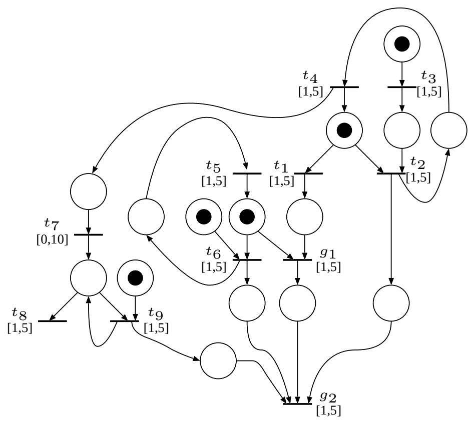


Fig. 15. Examples for the guided simulation.

noninterface transitions, and g_1 and g_2 are interface transitions, the necessary set of g_1 is $\{t_1\}$. **guided_sim_phase1** then computes a dependent set of each necessary transition. The dependent set of a transition t is a set of transitions whose firings may be necessary before t in order to avoid missing the possible concrete traces. For example, consider generating a concrete trace of the net shown in Figure 15 for an abstracted trace $g_1 g_2$. As shown above, the necessary set of g_1 is $\{t_1\}$. If t_1 is fired from the current marking, g_1 becomes enabled, but, g_2 can never be fired. In this case, firing t_3, t_2, t_4, and t_1 in this order leads to the correct concrete trace. It is, however, not easy to find such a correct firing sequence directly. Instead, our algorithm guarantees to generate the correct concrete trace by the backtracking mechanism with firing a sufficient set of transitions in each step. Such a sufficient set of transitions is the dependent set. It is formally stated that a dependent set of transition t is a set E of transitions, satisfying $t \in E$, and for each $x \in E$, the necessary transitions of *conflict*$(x) \cap$ *NonIF_Dum* are also in E. The dependent set can be defined as a closure, and so, the while loop in Figure 14 computes it. In our example, the dependent set of t_1 is $\{t_1, t_3\}$. Hence, even if **guided_sim_phase1** fires t_1 first, it eventually can fire t_3 and then t_2 after several backtrackings. After firing t_2, it is straightforward to fire t_4 and t_1, because they are the necessary transitions for g_1. When g_1 becomes enabled, its necessary set is $\{g_1\}$ and its dependent set is $\{g_1, t_6\}$. The correct transition to be fired here is t_6, and is again guaranteed to be found by the backtracking mechanism. The fired transitions are appended to the trace h, and it is returned when g becomes empty. Then, the trace h is passed to **guided_sim_phase2** in order to generate a regular trace based on h.

In **guided_sim_phase2**, each transition determined by **find_firing_trans** is fired until h, which is updated by also **find_firing_trans**, becomes empty as shown in Figure 16. In **find_firing_trans**, if the first transition t of h is interface, it implies that t is enabled in the current marking because all its necessary transitions are supposed to be fired. In order to satisfy the regularity, however, firable noninterface or dummy transitions that are concurrent with t should be fired before t, if they exist. This should be done carefully, if such a noninterface or dummy transition x is in conflict with some other transition.

```
guided_sim_phase2($h$, $G$, $V$, $x$, $\alpha$) {
    $f = null$;
    $fired = null$;
    while(true) {
        $(t, h)$ =find_firing_trans($\alpha$, $h$);
        $f$ = append($f$, $t$);
        $\alpha$ = fire($\alpha$, $t$);
        if ($h$ is empty) break;
    }
    return $f$;
}

find_firing_trans($\alpha$, $h$) {
    while(true) {
        $(t, h') =$ (head($h$), tail($h$));
        if ($t$ is interface)
            return find_concur_trans($\alpha$, $t$, $h$, $h'$);
        else
            if ($t \notin fired$) return $(t, h')$;
            else {
                $fired = fired - \{t\}$;
                $h = h'$;
            }
    }
}

find_concur_trans($\alpha$, $t$, $h$, $h'$) {
    find $x \in \mathsf{firable}(\alpha) \cap NonIE\_Dum$
            s.t. $conflict(x) \cap (\mathsf{prefix}(h, x) - fired) = \emptyset$;
    if (such $x$ exists) {
        $fired = fired \cup \{x\}$;
        return $(x, h)$;
    }
    else return $(t, h')$;
}
```

Fig. 16. Algorithm for guided simulation (2).

In such a case, an appropriate transition should be chosen such that it is consistent with the rest of h (up to the point where x fires). Let $\mathsf{prefix}(h, x)$ denote a prefix of h before the occurrence of x. It is equal to h if x is not included in h. Then, if $conflict(x) \cap \mathsf{prefix}(h, x) \neq \emptyset$, it implies that some other transition conflicting with x fires before x in h, and so, x should not be fired in the current timed state class. Thus, it is necessary to find a noninterface or dummy firable transition x such that $conflict(x) \cap \mathsf{prefix}(h, x) = \emptyset$. If such a x is found, it is returned with the unupdated h keeping the interface transition in its head. In this case, x is added to a global variable $fired$ to avoid firing it again when it is in the top of h. If such x does not exist, either there are no firable noninterface or dummy transitions, or every such transition conflicts with some interface transition in h. In either case, no firable noninterface or dummy transitions are concurrent with t. Hence, t and $h' = tail(h)$ are returned [5].

If t is noninterface or dummy, the regularity just requires that t should be fired. But, it may be already fired as mentioned above to generate regular traces when it is concurrent with some interface transition. Thus, if $t \in fired$, it is just removed from $fired$, and the next transition of h is processed. Otherwise, t and $h' = tail(h)$ are returned.

In the previous example, for an abstracted trace $g_1 g_2$,

$$t_3\ t_2\ t_4\ t_1\ t_6\ t_5\ g_1\ t_7\ t_9\ g_2$$

is obtained by **guided_sim_phase1**. This does not satisfy regularity, because t_7 and t_9 fire after g_1. When $h = g_1\ t_7\ t_9\ g_2$ is first given to **find_firing_trans**, (t_7, h) is returned with $fire = \{t_7\}$. After firing t_7, both t_8 and t_9 become firable. t_8 is not chosen, because $conflict(t_8) = \{t_9\}$ and $\mathsf{prefix}(h, t_8) = \{t_9\}$. After firing t_9, however, t_8 is chosen and fired. Finally, g_2 is fired, and the following regular trace is obtained.

$$t_3\ t_2\ t_4\ t_1\ t_6\ t_5\ t_7\ t_9\ t_8\ g_1\ g_2$$

Note that this second phase is deterministic (backtracking is not necessary), because every causal transition needed for interface transitions in g is found in the phase 1. An alternative of the guided simulation with only one phase may be possible, but we believe that the above approach minimizes the number of the backtrackings.

VII. Limitations

The proposed algorithm currently has the following restrictions on the class of timed STGs to be handled.

- Any loop in the given timed STG must contain at least one transition with non-zero delay (i.e., its earliest firing time is greater than 0). This restriction is necessary to guarantee the termination of the second phase of the guided simulation. In the untimed case, this termination is not guaranteed, if a loop formed only by noninterface or dummy transitions exists [23]. In the timed case, however, even if such a loop exists, time certainly passes in the loop from the above restriction, and eventually, other transitions are fired. Thus, the guided simulation can terminate.
- For every two reachable timed state classes of the STG, either one is reachable from the other. This restriction is necessary, because every CSC violation pair is found along a single path from the initial timed state class in our algorithm.

VIII. Experimental results

The proposed method has been naively implemented using the C language [6]. This section evaluates the potential performance of the proposed method and the area overhead of the synthesized circuits. The experiments here have been done on a 2.8 GHz Pentium 4 workstation with 4 gigabytes of memory.

For speed independent circuit synthesis, tools moebius [21] and csat [22] use the similar idea of the decomposition based synthesis. Since our method is extended for timed circuit synthesis and they run on different machines, the precise comparison with those tools is not very meaningful. According to the rough comparison on the several circuits used in [23] and [21], the performance of the synthesis and the

[5]If transitions conflicting with interface transitions form loops, then there can exist many nonequivalent concrete traces that correspond to g. In our current implementation, one shortest regular concrete trace is selected.

[6]The interleaving generation part in **find_essential** is not fully implemented currently.

TABLE I

EXPERIMENTAL RESULTS (1).

Circuit	Literal counts	
	atacs	Proposed
alloc-outbound	16	16
atod	9	9
chu150	11	11
chu172	6	6
converta	12	12
dff	8	8
master-read	26	26
mp-forward-pkt	16	16
nak-pa	20	20
nowick	18	18
pe-rcv-ifc	50	51
pe-send-ifc	60	61
ram-read-sbuf	20	20
rcv-setup	8	8
sbuf-ram-write	19	19
sbuf-read-ctl	13	13
sbuf-send-pkt2	19	19
sendr-done	5	5
trimos-send	21	21
vbe10b	21	21
vbe4a	6	6
vbe6a	17	17
vmebus-arb	9	9
wrdata	11	11
wrdatab	29	26
rappid	152	152

literal counts of the synthesized circuits do not make much difference.

For the timed circuit synthesis, in order to evaluate the area overhead of the proposed method, small timed specifications, which are obtained from the standard benchmarks by adding the fixed lower bound and upper bounds to each transition ([3,5] for output transitions, [8,10] for input transitions), are synthesized by the proposed method and a timed circuit synthesis tool atacs [24], and the literal counts of the synthesized circuits are compared. Table I except for the last line shows these results. These results show that the quality at least with respect to the area size is not badly affected, even though our method uses restricted information for synthesizing sub-circuits, and so may choose non-optimal input sets. Since these example are small, the CPU times for both methods are almost the same.

The last example shown in Table I is the control circuit for RAPPID. This example is larger, and so, atacs cannot complete the synthesis on the flat specification without hierarchical decomposition. The literal count shown for atacs in the table is obtained using hierarchical decomposition. The proposed method synthesizes it within 15 seconds.

Table II shows the results for much larger examples, which are taken from [14]. They are specifications for IIR filters, FIR filters, and portions of the Discrete Cosine Transform (DCT) circuits obtained from SpecC/Balsa high-level specifications (slightly modified versions are used for this experiment due to some improvement of our Balsa compiler). Those with "_b" are allowed to use more operational units. Thus, they have more concurrency than those with "_a". In order to evaluate the performance of the timed circuit synthesis, this table also shows the performance of synthesis of the untimed version (Speed Independent) of the above specifications. Note that the partial order reduction and POSET techniques are used for these examples, because otherwise the timed circuit synthesis does not terminate due to many dummy transitions left by the exact timed net contraction (the untimed circuit synthesis is not improved much by these techniques). Although the timed circuit synthesis takes longer time especially for the specifications with more concurrency, much more compact circuits (compared with the untimed versions) are successfully synthesized (by using the above techniques) without significant performance penalty.

TABLE II

COMPARISON BETWEEN UNTIMED AND TIMED CIRCUIT SYNTHESIS.

Circuits	Synthesis time (sec.)		Literal counts	
	Untimed	Timed	Untimed	Timed
IIR_a	13.9	18.1	449	391
IIR_b	19.2	65.0	567	462
FIR_a	215.5	268.0	1155	923
FIR_b	285.7	952.7	1604	1116
DCT_a	2952.0	3700.3	2207	1870
DCT_b	2752.2	3345.9	2496	1902

The untimed version of IIR_a is synthesized by atacs in about 200 seconds, but it runs out of memory for the other specifications. Since there are no hierarchy information for those designs, the hierarchal synthesis of atacs does not work. The only circuit synthesized by atacs from the untimed version of IIR_a has the same literal count 449 as the one synthesized by our method.

IX. CONCLUSION

This report presents a decomposition based method for efficient synthesis of large timed circuits. The idea proposed for the speed independent circuits [23] has been extended for the timed circuit synthesis. Since the state spaces of the original timed STGs are not needed to be explored, the proposed method allows for the synthesis of large timed circuits that could not be synthesized using conventional flat synthesis methods. Although this method does have some area overhead for small circuits, the experimental results show that the overhead appears to be very small.

ACKNOWLEDGMENT

We'd like to thank Walter Vogler and Mark Schaefer for their helpful comments. Furthermore, we would like to thank Josep Carmona, Jordi Cortadella, Victor Khomenko, and Alex Yakovlev for giving us their benchmarks and tools.

REFERENCES

[1] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. *IEICE Transactions on Information and Systems*, E80-D(3):315–325, March 1997.

[2] P. A. Beerel, C. J. Myers, and T. H.-Y. Meng. Covering conditions and algorithms for the synthesis of speed-independent circuits. *IEEE Transactions on Computer-Aided Design*, March 1998.

[3] R. M. Fuhrer, S. M. Nowick, M. Theobald, N. K. Jha, B. Lin, and L. Plana. Minimalist: An environment for the synthesis, verification and testability of burst-mode asynchronous machines. Technical Report TR CUCS-020-99, Columbia University, NY, July 1999.

[4] Steven M. Burns and Alain J. Martin. Syntax-directed translation of concurrent programs into self-timed circuits. In J. Allen and F. Leighton, editors, *Advanced Research in VLSI*, pages 35–50. MIT Press, 1988.

[5] Kees van Berkel, Joep Kessels, Marly Roncken, Ronald Saeijs, and Frits Schalij. The VLSI-programming language Tangram and its translation into handshake circuits. In *Proc. European Conference on Design Automation (EDAC)*, pages 384–389, 1991.

[6] Euiseok Kim, Jeong-Gun Lee, and Dong-Ik Lee. Automatic process-oriented control circuit generation for asynchronous high-level synthesis. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 104–113. IEEE Computer Society Press, April 2000.

[7] Joep Kessels and Ad Peeters. The Tangram framework: Asynchronous circuits for low power. In *Proc. of Asia and South Pacific Design Automation Conference*, pages 255–260, February 2001.

[8] Doug Edwards and Andrew Bardsley. Balsa: An asynchronous hardware synthesis language. *The Computer Journal*, 45(1):12–18, 2002.

[9] A. Bystrov and A. Yakovlev. Asynchronous circuit synthesis by direct mapping: Interfacing to environment. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 127–136, April 2002.

[10] Tiberiu Chelcea and Steven M. Nowick. Resynthesis and peephole transformations for the optimization of large-scale asynchronous systems. In *Proc. ACM/IEEE Design Automation Conference*, June 2002.

[11] Chris J. Myers and Teresa H.-Y. Meng. Synthesis of timed asynchronous circuits. *IEEE Transactions on VLSI Systems*, 1(2):106–119, June 1993.

[12] Tomohiro Yoneda and Chris Myers. Synthesizing timed circuits from high level specification languages. *NII Technical Report*, NII-2003-003E, 2003.

[13] A. Matsumoto. High level synthesis of asynchronous circuits (in Japanese). *Master Thesis, Tokyo Insitute of Technology*, 2004.

[14] T. Yoneda, A. Matsumoto, M. Kato, and C. Myers. High level synthesis of timed asynchronous circuits. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 178–189. IEEE Computer Society Press, 2005.

[15] N. Sretasereekul, H. Saito, M. Imai, E. Kim, M. Ozcan, K. Thongnoo, H. Nakamura, and T. Nanya. A zero-time-overhead asynchronous four-phase controller. *Proc. of ISCAS*, 2003.

[16] Tam-Anh Chu. *Synthesis of Self-Timed VLSI Circuits from Graph-Theoretic Specifications*. PhD thesis, MIT Laboratory for Computer Science, June 1987.

[17] Walter Vogler and Ralf Wollowski. Decomposition in asynchronous circuit design. In J. Cortadella, A. Yakovlev, and G. Rozenberg, editors, *Concurrency and Hardware Design*, volume 2549 of *Lecture Notes in Computer Science*, pages 152–190. Springer-Verlag, 2002.

[18] H. Zheng, E. Mercer, and C. J. Myers. Modular verification of timed circuits using automatic abstraction. *IEEE Transactions on Computer-Aided Design*, 22(9), September 2003.

[19] Ruchir Puri and Jun Gu. A modular partitioning approach for asynchronous circuit synthesis. In *Proc. ACM/IEEE Design Automation Conference*, pages 63–69, June 1994.

[20] J. Beister, G. Eckstein, and R. Wollowski. From STG to extended-burst-mode machines. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 145–158, April 1999.

[21] J. Carmona and J. Cortadella. ILP models for the synthesis of asynchronous control circuits. *Proc. of the IEEE/ACM International Conference on Computer Aided Design*, pages 818–825, 2003.

[22] A. Yakovlev V. Khomenko, M. Koutny. Logic synthesis for asynchronous circuits based on petri net unfoldings and incremental sat. *Proc. of ACSD*, 2004.

[23] T. Yoneda, H. Onda, and C. Myers. Synthesis of speed independent circuits based on decomposition. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 135–145. IEEE Computer Society Press, April 2004.

[24] Chris J. Myers. *Computer-Aided Synthesis and Verification of Gate-Level Timed Circuits*. PhD thesis, Dept. of Elec. Eng., Stanford University, October 1995.

[25] D. L. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits*. MIT press, 1988.

[26] B. Berthomieu and M. Diaz. Modeling and verification of time dependent systems using time Petri nets. *IEEE Trans. on Software Eng.*, 17(3):259–273, 1991.

[27] T. Yoneda and H. Schlingloff. Efficient verification of parallel real-time systems. *Formal Method in System Design*, pages 187–215, 1997.

[28] T. Yoneda, E. G. Mercer, and C. J. Myers. Modular synthesis of timed circuits using partial order reduction. *Proc. of The 10th Workshop on Synthesis And System Integration of Mixed Technologies*, pages 127–134, 2001.

[29] E. G. Mercer, C. J. Myers, and T. Yoneda. Improved POSET timing analysis in timed Petri nets. *Proc. of The 10th Workshop on Synthesis And System Integration of Mixed Technologies*, pages 151–158, 2001.

[30] Sung Tae Jung and Chris J. Myers. Direct synthesis of timed circuits from free-choice STGs. *IEEE Transactions on Computer-Aided Design*, 21(3):275–290, March 2002.

[31] Kenneth McMillan. Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In G. v. Bochman and D. K. Probst, editors, *Proc. International Workshop on Computer Aided Verification*, volume 663 of *Lecture Notes in Computer Science*, pages 164–177. Springer-Verlag, 1992.