

Modular Synthesis of Timed Circuits using Partial Order Reduction

Tomohiro Yoneda*

Department of Computer Science
Tokyo Institute of Technology
Tokyo, Japan 152-8552
yoneda@cs.titech.ac.jp

Eric Mercer Chris Myers†

Department of Electrical Engineering
University of Utah
Salt Lake City, UT, U.S.A. 84112
myers@ee.utah.edu

Abstract — This paper develops a modular synthesis algorithm for timed circuits that is dramatically accelerated by partial order reduction. This algorithm synthesizes each module in a hierarchical design individually. It utilizes partial order reduction to reduce the state space explored for the other modules by considering a single interleaving of concurrently enabled transitions. This approach better manages the state explosion problem resulting in a more than 2 order of magnitude reduction in synthesis time. The improved synthesis time enables the synthesis of a larger class of timed circuits than was previously possible.

I. INTRODUCTION

In order to satisfy the requirements of high performance, designers are trying to implement aggressive and complicated timed circuits. However, designing those circuits in a fully manual style is very difficult. Assistance of CAD tools is essential for synthesis and verification. One important issue in developing such CAD tools is how to avoid the state explosion problem.

In the area of formal verification, a technique called *partial order reduction* has been investigated for some time [1, 2, 3, 4, 5, and others]. Based on the fact that all the possible interleavings of concurrent events are not relevant for the properties to be checked, this technique allows us to do the verification in the reduced state spaces efficiently. Partial order reduction is extremely important because approaches based on decision diagrams do not work well for the verification of timed systems.

Generating a reduced state space for the synthesis of timed circuits using partial order reductions is seemingly inapplicable. This is because a complete state space is required to synthesize a flat circuit. Work in [6] introduces a POSET method that applies a partial order reduction in the timing representation of a timed state space, but it still explores all reachable states of the specification to satisfy the synthesis requirements. Works in [7, and others] show that the net unfolding technique can be used for synthesis of untimed circuits without the whole explicit state space. However, it is not clear that these methods

can be extended efficiently to timed circuit synthesis.

Recent work in [8] presents a modular verification approach that verifies each subcircuit of a hierarchical design individually by conservatively abstracting away signals not on the interface of the target subcircuit. These abstracted signals are invisible to the target subcircuit, and thus can be handled differently to reduce the size of the explored state space. This modular abstraction approach is extended to the synthesis problem in [9]. Experimental results in [8] and [9] demonstrate that the modular abstraction approach can handle substantially larger circuits than other approaches that ignore hierarchy in the specification. However, because the modular abstraction approach generates a conservative approximation of timing behaviors, it is possible for synthesis to generate non-optimal subcircuit designs. The work presented in this paper addresses this issue by reducing the explored timed state space in the synthesis problem without approximating timing behaviors.

The work in this paper takes advantage of hierarchy in a specification by applying partial order reductions to modular synthesis. The approach is similar to modular synthesis using abstraction, however, rather than abstracting away signals that are invisible to the target subcircuit, this paper proposes to consider a single interleaving of those signals. This approach always generates optimal circuits because exact timing behavior at the subcircuit interface is preserved by the partial order reduction. Furthermore, this approach can potentially be applied with the abstraction method to yield a further reduction in the explored state space. This paper demonstrates that the new approach reduces the state explosion problem in synthesis yielding a more than 2 order of magnitude reduction in synthesis time when compared to approaches that ignore hierarchy.

The rest of this paper is organized as follows. Section 2 reviews time Petri nets, which are the formal model of our approach, and their related definitions. Section 3 formalizes modular synthesis, and explains it with a simple example. Section 4 describes our main idea, the partial order reduction for synthesis, as well as the proofs to show that it is an exact method. The performance of the proposed method is demonstrated in Section 5 using STARI circuits. Finally, Section 6 summarizes our work.

*This research is supported by JSPS Joint Research Projects.

†This research is supported by NSF CAREER award MIP-9625014, NSF Japan Program award INT-0087281, and SRC grant 99-TJ-694.

II. TIME PETRI NETS

A *time Petri net* N is a six-tuple, $N = (P, T, F, \text{Eft}, \text{Lft}, \mu_0)$, 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*, $\text{Eft} : T \rightarrow \mathbf{Q}^+$, $\text{Lft} : T \rightarrow \mathbf{Q}^+ \cup \{\infty\}$ are functions for the *earliest* and *latest firing times* of transitions satisfying $\text{Eft}(t) \leq \text{Lft}(t)$ for all $t \in T$ (\mathbf{Q}^+ denotes the set of nonnegative rationals), and $\mu_0 \subseteq P$ is the *initial marking* of the net.

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 , respectively.

A *marking* μ of N 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 $\text{enabled}(\mu)$ be the set of transitions enabled in μ . A (timed) *state* σ of a time Petri net is a pair (μ, clock) , where μ is a marking and clock is a function $T \rightarrow \mathbf{Q}^+$. The *initial state* σ_0 is (μ_0, clock_0) , where $\text{clock}_0(t) = 0$ for all $t \in T$. The states of a time Petri net change if time passes or if a transition fires. In state $\sigma = (\mu, \text{clock})$, time $\tau \in \mathbf{Q}^+$ can pass, if for all $t \in \text{enabled}(\mu)$, $\text{clock}(t) + \tau \leq \text{Lft}(t)$. In this case, state $\sigma' = (\mu', \text{clock}')$ is obtained from σ by passing τ , if

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

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

1. $\mu' = (\mu - \bullet t_f) \cup t_f\bullet$, and
2. for all $t \in T$,

$$\text{clock}'(t) = \begin{cases} 0 & \text{if } t \in \text{enabled}(\mu') - \\ & \text{enabled}(\mu - \bullet t_f), \\ \text{clock}(t) & \text{else.} \end{cases}$$

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. Our method assumes that there is no loop structure in which every transition has 0 as its earliest firing time.

A *module* is a tuple (I, O, N, wire) , where I is a set of input wires, O is a set of output wires, $N = (P, T, F, \text{Eft}, \text{Lft}, \mu_0)$ is a time Petri net, and $\text{wire} : T \rightarrow (I \cup O) \times \{+, -\}$. For simplicity, this paper also uses $\text{wirename}(t)$ to represent the wire name only, i.e., if $\text{wire}(t) = u+$, then $\text{wirename}(t) = u$. If $\text{wirename}(t) \in I$, t is called an *input transition*. Similarly, if $\text{wirename}(t) \in O$, it is called an *output transition*. This paper assumes that a firing of a transition t causes a change of value in the wire $\text{wirename}(t)$, and its direction ($0 \rightarrow 1$ or $1 \rightarrow 0$) is represented by $+$ or $-$. Our method uses modules to represent specifications of subcircuits or behavior of environments.

Our method considers a set $\{M_1, M_2, \dots, M_n\}$ of modules, only if for any i and j such that $i \neq j$, $O_i \cap O_j = \emptyset$, $P_i \cap P_j = \emptyset$, and $T_i \cap T_j = \emptyset$, where $M_k = (I_k, O_k, N_k, \text{wire}_k)$ and $N_k = (P_k, T_k, F_k, \text{Eft}_k, \text{Lft}_k, \mu_{0k})$ for $1 \leq k \leq n$. If the set satisfies $\bigcup_{i=1}^n O_i \supseteq \bigcup_{i=1}^n I_i$, then the system is *closed*. Since the transition sets of any two modules are disjoint, this paper uses *wire* or *wirename* in every module instead of wire_k or wirename_k .

In a set of modules, an output transition fires according to the firing rules of time Petri nets. However, an input transition fires only in synchronization with the corresponding output transition in some different module. More precisely, when an output transition t_{out} fires, every module whose input wire set includes $\text{wire}(t_{out})$ must have one input transition t_{in} with $\text{wire}(t_{in}) = \text{wire}(t_{out})$ that can fire at the same moment as t_{out} .

In order to handle the possible infinite behaviors of a time Petri net, we use a *timed state class* that is a finite representation of a set of (infinitely) many (timed) states. A timed state class is $s = (\mu, I)$, where μ is a marking and I is a set of inequalities. Intuitively, in a state σ , if a transition t can fire after passing some time τ , and a new state σ' is obtained, then there exists a timed state class transition $s \xrightarrow{t} s'$ such that $\sigma \in s$ and $\sigma' \in s'$, and vice versa. The details about timed state classes can be found in [10] and [11], although some different terminology is used. When considering a set of modules, μ and I of a timed state class are the union of the markings and the inequality sets respectively for each of the time Petri nets in the module set. When our method obtains the timed state classes s_1, s_2, \dots by firing output transitions t_1, t_2, \dots from s_0 , it is represented by $s_0 \xrightarrow{t_1} s_1 \xrightarrow{t_2} \dots$, which is called a *timed state class sequence*, or a TSCS for short. We write $s \xrightarrow{t} s' \in \rho$, if the timed state class transition $s \xrightarrow{t} s'$ is included in a TSCS ρ (i.e., $\rho = s_0 \xrightarrow{t_1} s_1 \xrightarrow{t_2} \dots s \xrightarrow{t} s' \dots$). This notation is also used for a set of TSCSs.

III. SYNTHESIS OF TIMED CIRCUITS

Let us consider a set $M = \{E, S_1, \dots, S_i, \dots, S_n\}$ of modules which is a closed system, where E is the environment of the whole circuit and S_i is a specification of the i -th subcircuit. Let $S_i = (I_i, O_i, N_i, \text{wire})$ and $N_i = (P_i, T_i, F_i, \text{Eft}_i, \text{Lft}_i, \mu_{0i})$. The goal of modular synthesis is to synthesize a subcircuit specified by S_i . Note that our method expects that the synthesized subcircuit works correctly with respect to S_i in M . It may, however, work incorrectly with respect to S_i in a different module set. Thus, the synthesis procedure depends on the behavior of the other modules E, S_1, \dots, S_n .

For a set \mathcal{T} of TSCSs and a module S_i , we define the following.

$$\begin{aligned} \text{visible}(\mathcal{T}, S_i) = \{ & (m, m') \mid \text{wirename}(t) \in I_i \cup O_i, \\ & s \xrightarrow{t} s' \in \mathcal{T}, s = (\mu, I), s' = (\mu', I'), \\ & m = \mu \cap P_i, m' = \mu' \cap P_i \} \end{aligned}$$

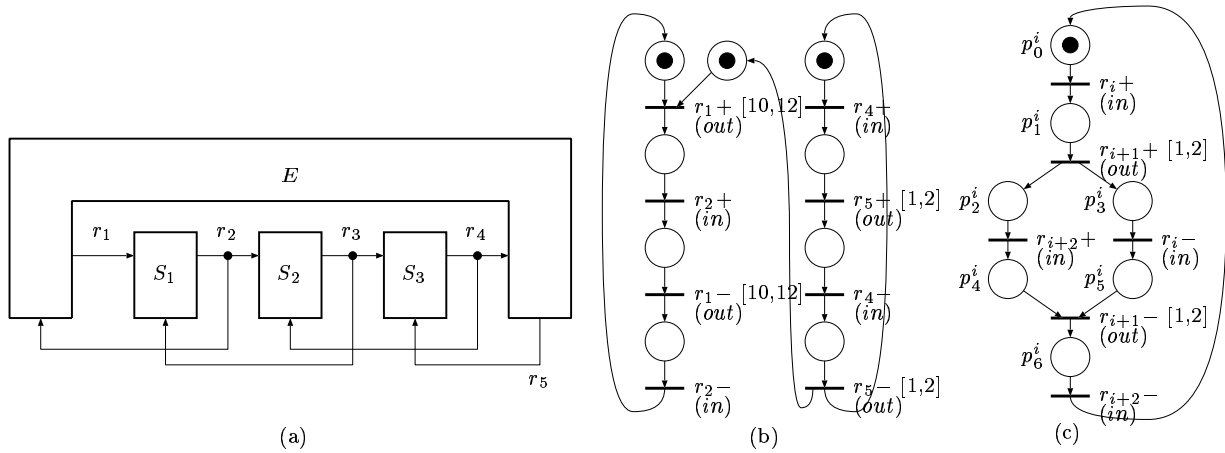


Fig. 1. (a) a simple example, (b) the environment E , and (c) the module being synthesized S_i .

where m and m' are markings of N_i . This represents the set of marking transitions (projected to P_i) that can occur in \mathcal{T} changing the values of wires in S_i . Let (u_1, u_2, \dots, u_l) be an ordered set of input and output wires of S_i (i.e., for $1 \leq k \leq l$, $u_k \in I_i \cup O_i$, and $l = |I_i \cup O_i|$). $\text{val}(u_k, m)$ denotes the value of wire u_k in S_i when the marking of S_i is m . $\text{st}(m, S_i)$ denotes a vector (x_1, x_2, \dots, x_l) for the values of input and output wires in S_i , where

$$x_k = \begin{cases} R & \text{if } t \in \text{enabled}(m) \text{ such that} \\ & \text{wire}(t) = u_k+ \text{ and } u_k \in O_i \\ F & \text{if } t \in \text{enabled}(m) \text{ such that} \\ & \text{wire}(t) = u_k- \text{ and } u_k \in O_i \\ \text{val}(u_k, m) & \text{else.} \end{cases}$$

Thus, $\text{st}(m, S_i)$ represents a state vector of S_i that also indicates the excited output wires by R (rising) and F (falling). A *reduced state graph* is defined as follows:

$$\text{rsg}(\mathcal{T}, S_i) = \{ \{v \rightarrow v'\} \mid (m, m') \in \text{visible}(\mathcal{T}, S_i), \\ v = \text{st}(m, S_i), v' = \text{st}(m', S_i) \}$$

$\text{rsg}(\mathcal{T}, S_i)$ represents the transition relation between state vectors of S_i which is defined by \mathcal{T} . Thus, $\text{rsg}(\mathcal{T}, S_i)$ specifies a state graph of S_i .

Let $\mathcal{T}(M)$ denote the set of TSCSs of M such that every TSCS starts from the initial timed state class s_0 of M . A modular synthesis with respect to S_i in M is done by applying a synthesis algorithm for timed circuits in [12], denoted by *synthesis*, to $\text{rsg}(\mathcal{T}(M), S_i)$. Hence, $C_i = \text{synthesis}(\text{rsg}(\mathcal{T}(M), S_i))$ is our goal.

Fig.1(a) shows an example of a set of modules, and (b) and (c) are the time Petri nets for E and S_i , respectively. Since input transitions cannot fire by themselves, the earliest and latest firing times for them are not shown in the figure. Formally, they have $[0, \infty]$ bounds. E simply sets r_1 and waits for r_2 to rise, and then resets r_1 . Concurrently, E waits for r_4 to rise and then sets r_5 followed by the resetting phase. E again raises r_1

when both r_2 and r_5 fall. When r_i rises, S_i sets r_{i+1} and waits for both r_{i+2} to rise and r_i to fall. Then, S_i resets r_{i+1} and waits for r_{i+2} to be reset. Thus, it has a TSCS like $s_0 \xrightarrow{r_1+} s_1 \xrightarrow{r_2+} s_2 \xrightarrow{r_3+} s_3 \xrightarrow{r_4+} s_4 \xrightarrow{r_5+} \dots$. Note that for simplicity, r_1+ , r_2+ , \dots are used instead of their real transition names in this example. Let us consider the synthesis of the subcircuit $S_2 = (\{r_2, r_4\}, \{r_3\}, N_2, \text{wire})$. Since only r_2 , r_3 , and r_4 change the markings of N_2 , we have ¹

$$\text{visible}(\mathcal{T}(M), S_2) = \{ (\{p_0\}, \{p_1\}), (\{p_1\}, \{p_2, p_3\}), \\ (\{p_2, p_3\}, \{p_4, p_3\}), (\{p_4, p_3\}, \{p_4, p_5\}), \\ (\{p_4, p_5\}, \{p_6\}), (\{p_6\}, \{p_0\}) \}.$$

Initially, all wires take the value 0. Thus, the method finds the state graph of S_i , specified by $\text{rsg}(\mathcal{T}(M), S_2)$, as shown in Fig.2 by the unshaded states. Since only the subcircuit with output r_3 is being synthesized, only the excitation information (such as R and F) with respect to r_3 is indicated in the state graph. Once a state graph is obtained, an actual circuit is synthesized by a timed circuit synthesis method. In this case, r_3 is produced by a simple buffer with r_2 in its input. The marking pair $(\{p_2, p_5\}, \{p_4, p_5\})$ is not included in $\text{visible}(\mathcal{T}(M), S_2)$ because the environment E changes r_1 very slowly compared to other outputs. Thus, r_2- always occurs later than r_4+ . The shaded state in Fig.2 represents this marking pair that is omitted due to the timing specification of E . A traditional speed independent approach ignores timing information and would include the shaded state of Fig.2. This extra state forces the use of a Muller C-element to produce r_3 . The resulting circuit is larger and slower than the simple buffer produced by timed circuit synthesis. A key advantage of timed circuit synthesis is this potential to generate smaller faster circuits.

¹The superscripts indicating the i -th specification are omitted in the place names for simplicity.

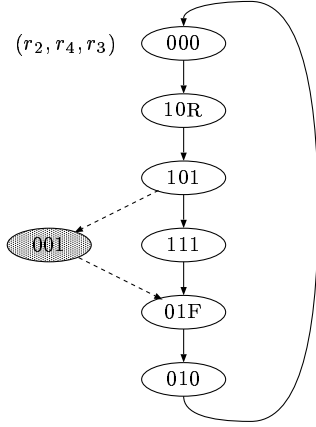


Fig. 2. State graph of S_2 .

If we consider $\{S_1, \dots, S_n\}$ to be one module and assume that there are no internal signals in the environment, it is equivalent to the non-modular synthesis method. In modular synthesis, the state graphs (i.e., $\text{rsg}(\mathcal{T}(M), S_i)$) are much smaller than those for non-modular synthesis, because many variables are projected out in $\text{rsg}(\mathcal{T}(M), S_i)$. In order to obtain $\mathcal{T}(M)$, however, this method has to generate the full state space of the whole system M , even if one subcircuit S_i is being considered at a time. Thus, as long as this method uses $\mathcal{T}(M)$, there is no advantage to modular synthesis. Work in [9] proposes to use a reduced state space $\mathcal{T}'(M)$ by means of an automated abstraction technique. In that case, the run time and memory space for synthesizing a subcircuit are reduced. Although $\mathcal{T}'(M)$ needs to be computed for each subcircuit, the total run time for synthesizing all subcircuits is much shorter than the non-modular method. The advantage with respect to maximum required memory space in modular synthesis is obvious. The problem with this method is that not all invisible behavior can be easily abstracted, and the abstractions can result in additional timing behavior leading to non-optimal circuit realizations. This paper proposes another approach to generate a reduced state space $\mathcal{T}_{S_i}(M)$ by means of partial order reduction. The advantages of the partial order approach are two-fold: first, concurrent interleavings of invisible behavior are not considered; and second, an optimal circuit realization is always obtained. The disadvantage of the partial order approach is that the invisible behavior still needs to be considered. Note that it is possible to apply the abstraction technique and then to apply the partial order reduction.

IV. PARTIAL ORDER REDUCTION

$\mathcal{T}(M)$ is obtained by firing every possible output transition from every reachable timed state class. This *total order exploration* algorithm often suffers from the state explosion problem. A *partial order exploration* algorithm

generates a set of reduced TSCSs with respect to S_i that still has sufficient information to synthesize a correct circuit for S_i .

In order to explain the proposed idea more formally, this section first defines *untimed-project*. For a TSCS $s \xrightarrow{t} s' \xrightarrow{t'} \dots$ with $s = (\mu, I)$ and a specification S_i ,

$$\text{untimed-project}(s \xrightarrow{t} s' \xrightarrow{t'} \dots, S_i) = \begin{cases} m \xrightarrow{t} Y & \text{if wirename}(t) \in I_i \cup O_i, \\ Y & \text{else} \end{cases},$$

where $Y = \text{untimed-project}(s' \xrightarrow{t'} \dots, S_i)$, and $m = \mu \cap P_i$. This definition can also be extended for a set of TSCSs.

Now, for a given set M of modules and a module S_i for the specification of the target subcircuit, our method constructs a set $\mathcal{T}_{S_i}(M)$ of reduced TSCSs such that the following property holds.

Property 1

$$\text{untimed-project}(\mathcal{T}(M), S_i) = \text{untimed-project}(\mathcal{T}_{S_i}(M), S_i).$$

If this property holds, then the following lemma and theorem can be proven as follows.

Lemma 1 The following relation holds.

$$\text{visible}(\mathcal{T}(M), S_i) = \text{visible}(\mathcal{T}_{S_i}(M), S_i).$$

Proof: Suppose $(m, m') \in \text{visible}(\mathcal{T}(M), S_i)$. Then, there exists a TSCS $\rho = s_0 \xrightarrow{t_1} \dots s \xrightarrow{t} s' \dots$ in $\mathcal{T}(M)$ such that $\text{wirename}(t) \in I_i \cup O_i$, $m = \mu \cap P_i$, and $m' = \mu' \cap P_i$, where $s = (\mu, I)$ and $s' = (\mu', I')$. From Property 1, there also exists a TSCS ρ' in $\mathcal{T}_{S_i}(M)$ such that $\text{untimed-project}(\rho, S_i) = \text{untimed-project}(\rho', S_i)$. This implies $(m, m') \in \text{visible}(\mathcal{T}_{S_i}(M), S_i)$. The other direction can be proven similarly. \square

Theorem 1 The following relation holds.

$$\text{rsg}(\mathcal{T}(M), S_i) = \text{rsg}(\mathcal{T}_{S_i}(M), S_i).$$

Proof: From Lemma 1, we have the same sets of pairs (m, m') for both $\mathcal{T}(M)$ and $\mathcal{T}_{S_i}(M)$. Thus, we have the same sets of $\langle v \rightarrow v' \rangle$ for them. Hence, we have $\text{rsg}(\mathcal{T}(M), S_i) = \text{rsg}(\mathcal{T}_{S_i}(M), S_i)$. \square

This theorem shows that the proposed method gives not an approximation but an exact solution.

Next, this section shows how to construct $\mathcal{T}_{S_i}(M)$ such that Property 1 holds. The idea of partial order reduction is to prune some successor timed state classes during state space traversal. This means that some firing orders between transitions are omitted. However, if some firing orders between transitions in T_i are missed, then obviously Property 1 does not hold. Thus, our method has to guarantee that every possible firing order between transitions in T_i is generated. In addition, suppose that t_1 and t_2 which are outside S_i are in *conflict* (i.e., only one

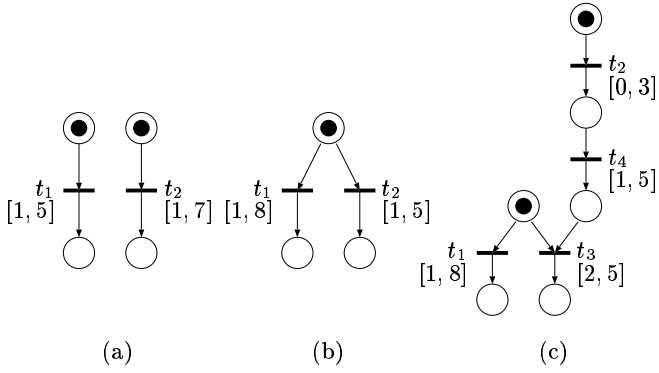


Fig. 3. Concurrent and conflicting transitions.

of t_1 or t_2 can occur). If the firing of t_1 , for example, is missed in $\mathcal{T}_{S_i}(M)$, then the behavior of S_i that is caused by the descendant of t_1 may be missed too. This results in a wrong subcircuit. Thus, for η in $\mathcal{T}(M)$, we have to guarantee that $\mathcal{T}_{S_i}(M)$ includes η' such that the same set of transitions eventually fire in both η and η' .

This latter requirement is also necessary for verification. Thus, we first explain how the partial order reduction algorithm for verification constructs reduced state spaces. Basically, it generates only one firing order for concurrent transitions, and generates every possible firing order for conflicting transitions. For example, for the time Petri net shown in Fig.3(a), it fires either t_1 or t_2 from the current timed state class, and for the one in the Fig.3(b), it fires both t_1 and t_2 . However, it is not so simple. Consider the time Petri net shown in the Fig.3(c). In this case, t_3 is not enabled currently, and so, it is not possible to fire both t_1 and t_3 . However, the firing sequence starting from t_2 can make t_3 ready to fire if the firing of t_1 is postponed. Thus, if t_1 is fired alone, then the possibility of the firing of t_3 is missed. Therefore, our method cannot prune the successor timed state class obtained by t_2 even if t_1 and t_2 are concurrent. In order to handle general cases, if our method wants to fire a transition t at a timed state class s , it must compute $\text{dependent}(s, t)$ which is a set of enabled output transitions such that the interleavings of the firings of those transitions should be generated for the correct results. For example, $\text{dependent}(s, t_1) = \{t_1\}$ for Fig.3(a), and $\text{dependent}(s, t_1) = \{t_1, t_2\}$ for Fig.3(b) and (c).

The above algorithm can be modified to satisfy the following 2 requirements that are necessary for property 1 to hold: 1) all firing orders of T_i are explored; and 2) all firing orders of conflicting transitions are explored. Since the above algorithm tries to generate every possible firing order between conflicting transitions, our new algorithm for modular synthesis treats the transitions in T_i as being in conflict. That is, this method uses the following set $\text{conflict}(t)$ as the set of transitions which are in conflict with t .

$$\text{conflict}(t) = \begin{cases} T_i & \text{if } t \in T_i \\ \{t' \mid \bullet t \cap \bullet t' \neq \emptyset\} & \text{else.} \end{cases}$$

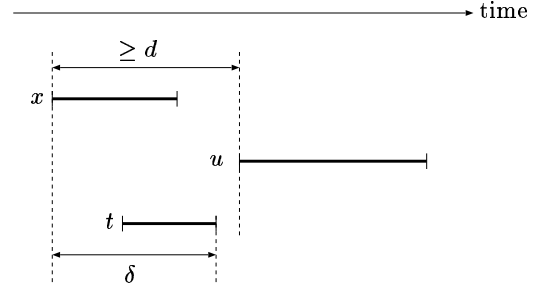


Fig. 4. Time separations between transitions.

The rest of the algorithm remains unchanged as explained below. The details can be found in [11].

For an output transition t enabled in a timed state class s , let $\text{sync_trans}(s, t)$ denote the set of enabled transitions that fire synchronously with t . That is, all transitions in $\text{sync_trans}(s, t)$ are input transitions except for t , and for any $t' \in \text{sync_trans}(s, t)$, t' is enabled in s and $\text{wire}(t') = \text{wire}(t)$ holds. An output transition t is called *fireable* in a timed state class s , if for some timed state class s' , $s \xrightarrow{t} s' \in \mathcal{T}(M)$. For a fireable output transition t and a timed state class s , $\text{dependent}(s, t)$ must include t and satisfy the following.

$$\begin{aligned} &\text{if } t_1 \in \text{dependent}(s, t), \text{ then} \\ &\quad \text{for every } u \in \bigcup_{t' \in \text{sync_trans}(s, t_1)} \text{conflict}(t'), \\ &\quad \quad \text{active}(s, u, t_1) \subseteq \text{dependent}(s, t), \end{aligned}$$

where $\text{active}(s, u, t)$ is the set of output transitions whose firings possibly lead the time Petri net to a timed state class where u is enabled in time in the sense that u can fire earlier than t . If u cannot get enabled in time, then our method does not have to consider the interleavings of the firings of t and u . More formally,

$$\text{active}(s, u, t) = \{x \mid (x, d) \in \text{necessary}(s, u, \{t\}), \\ x \text{ can fire } d \text{ time units earlier than } t\},$$

where $\text{necessary}(s, u, T_D)$ is a set of enabled output transitions (with some timing information) such that in order to fire u , the firing of at least one transition in this set is necessary. Intuitively, the computation of $\text{necessary}(s, u, T_D)$ is implemented such that empty source places of output transitions are searched upwards from u until some enabled output transition is found. For example, when our method computes $\text{necessary}(s, t_3, T_D)$ in Fig.3(c), this search is continued until t_2 is found. During this process, if an input transition is reached, the upward search is restarted from the corresponding output transition, because input transitions cannot control their firings. T_D is used to terminate loops. $\text{necessary}(s, t, T_D)$ is actually the set of pairs (x, d) where x is a transition found in the above search process, and d is the sum of the earliest firing times in the shortest path from x to u (with $\text{Eft}(x)$ not included and $\text{Eft}(u)$ included). This property of d implies that it takes at least d time units for u to become

ready to fire after the firing of x (see Fig.4). Therefore, if the maximum time separation δ between x and t is less than d time units, it means that u can never fire before t as shown in Fig.4. In other words, our method does not have to consider the interleavings between t and u in this case. Hence, $\text{active}(s, u, t)$ only contains those x which can fire d time units earlier than t . For example, $\text{necessary}(s, t_3, \{t_1\}) = \{(t_2, 3)\}$ in Fig.3(c), and so t_2 is in the active set. If the latest firing time of t_1 is 2, there is no possibility for t_3 to fire, and hence t_2 is not included in the active set.

Finally, our method constructs $\mathcal{T}_{S_i}(M)$ by firing only transitions in $\text{ready}(s)$ in each s , where $\text{ready}(s)$ is dependent(s, t) for some output transition t fireable in s , such that all transitions in the set are fireable.

Now, we have the following lemma.

Lemma 2 Property 1 holds.

Sketch of Proof: In this sketch, we assume that multiple firings of the same transitions can be distinguished by some appropriate way, and we simply use transition names here. For a more formal proof, the technique shown in [10] can be used.

(i) We first show that for any TSCS η in $\mathcal{T}(M)$, there exists a TSCS η' in $\mathcal{T}_{S_i}(M)$ such that $\text{untimed-project}(\eta, S_i) = \text{untimed-project}(\eta', S_i)$. To do this, we define several notations. For a transition t that fires in a TSCS η , let η^t be the prefix of η up to t , and $\text{Vis}(\eta^t)$ be the set of transitions in $\text{outtrans}(I_i \cup O_i)$ appearing in η^t , where $\text{outtrans}(I_i \cup O_i)$ is a set of output transitions u such that $\text{wirename}(u) \in I_i \cup O_i$. Furthermore, we define $\text{cong}(t, \eta)$, which is a subset of TSCSs in $\mathcal{T}_{S_i}(M)$, satisfying, for each $\gamma \in \text{cong}(t, \eta)$,

- every transition in η^t also fires in γ (maybe in a different order), and
- the transitions in $\text{Vis}(\eta^t)$ fire in exactly the same order in both η^t and γ , although in γ the transitions in $\text{outtrans}(I_i \cup O_i) - \text{Vis}(\eta^t)$ fire only after those in $\text{Vis}(\eta^t)$.

We show inductively that such $\text{cong}(t, \eta)$ is nonempty for any t in η . Assume that for some t in η , we already have $\text{cong}(t, \eta)$ satisfying the above properties, and suppose the next transition in η is t' . Here, we only consider the case that t' is in $\text{outtrans}(I_i \cup O_i)$, because the remaining case can be proven similarly. Let Γ_a be a subset of $\text{cong}(t, \eta)$ such that each element of Γ_a includes a transition t'_c with $\bullet t'_c \cap \bullet t' \neq \emptyset$ (i.e., t'_c which is in conflict with t' fires in the element). Γ_b is a subset of $\text{cong}(t, \eta) - \Gamma_a$ such that each element γ of Γ_b satisfies $\text{Vis}(\gamma^{t'}) - \text{Vis}(\eta^t) - \{t'\} \neq \emptyset$ (i.e., $t' \in \text{outtrans}(I_i \cup O_i)$ fires in η , while some different transition in $\text{outtrans}(I_i \cup O_i)$ fires before t' in elements of Γ_b). Γ_c is equal to $\text{cong}(t, \eta) - \Gamma_a - \Gamma_b$. In order to construct $\text{cong}(t', \eta)$, we discard Γ_a and Γ_b because it's impossible to fire t' from them in the correct positions, and $\text{cong}(t', \eta)$ is equal to Γ_c . Here, we should note the following two points. First, from the definition of $\text{conflict}(t)$ and the construction of the ready set, it's not possible that every element

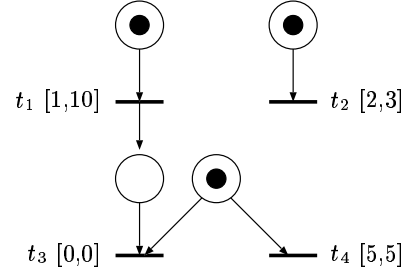


Fig. 5. An example of a time Petri net.

of $\text{cong}(t, \eta)$ is classified into either Γ_a or Γ_b . For example, if $\gamma \in \Gamma_b$ includes $t'_c \in \text{outtrans}(I_i \cup O_i) - \text{Vis}(\eta^t) - \{t'\}$, then the necessary set for t' should be included in the ready set and TSCSs obtained by firing those transitions are included in Γ_c . Second, t' occurs in all TSCSs in Γ_c , because t' is enabled from the first property of $\text{cong}(t, \eta)$ and time certainly passes from the assumption that there is no loop of transitions with $\text{Eft}(t) = 0$. From these considerations, we can say that $\text{cong}(t', \eta)$ is nonempty. Thus, the proof for the induction step is done. We can do the proof for the base step also similarly but very easily, because in this step, we only have to consider Γ_c above. Now, we have shown that for any $\eta \in \mathcal{T}(M)$ and any t in η , we can construct a nonempty set $\text{cong}(t, \eta)$. This implies that there exists a TSCS $\eta' \in \mathcal{T}_{S_i}(M)$ such that $\text{untimed-project}(\eta, S_i) = \text{untimed-project}(\eta', S_i)$.

(ii) Next, we show that for any TSCS $\eta \in \mathcal{T}_{S_i}(M)$, there exists a TSCS $\eta' \in \mathcal{T}(M)$ such that $\text{untimed-project}(\eta, S_i) = \text{untimed-project}(\eta', S_i)$. Since every transition in the ready sets is fireable, there exists $\eta' \in \mathcal{T}(M)$ that is obtained from η by permuting only concurrent transitions outside S_i (i.e., transitions which are not included in the ready set at the same time) in η . For example, in Fig.5, suppose that every transition is an output transition not in $\text{outtrans}(I_i \cup O_i)$, and that every token in the figure is produced at the same time τ_0 . Then, the partial order exploration algorithm may generate a TSCS $s_0 \xrightarrow{t_1} s_1 \xrightarrow{t_2} s_2 \xrightarrow{t_4} s_3$, where $\text{ready}(s_0) = \{t_1\}$, $\text{ready}(s_1) = \{t_2\}$, and $\text{ready}(s_2) = \{t_3, t_4\}$. On the other hand, the total order exploration algorithm gives the timing constraint like $v(t_1) \leq v(t_2)$ when firing t_1 before t_2 , where $v(t)$ is a variable representing the firing time of a transition t . From $v(t_3) = v(t_1) \leq v(t_2) \leq \tau_0 + 3$ and $\tau_0 + 5 \leq v(t_4)$, t_3 must fire before t_4 , and so, the above TSCS cannot be produced by the total order exploration algorithm. However, this situation occurs only for concurrent transitions. Since the partial order exploration algorithm assumes that all transitions in $\text{outtrans}(I_i \cup O_i)$ are in conflict even though they are structurally concurrent, it generates only firing orders which have correct timings for those transitions (i.e., if every transition in Fig.5 is in T_i , $s_0 \xrightarrow{t_1} s_1 \xrightarrow{t_2} s_2 \xrightarrow{t_4} s_3$ is not generated by the partial order exploration algorithm). Thus, the firing order between conflicting transitions or transi-

tions in T_i is identical in η and η' . Hence, for such η' , $\text{untimed-project}(\eta, S_i) = \text{untimed-project}(\eta', S_i)$ holds. \square

V. EXPERIMENTAL RESULTS

In order to show the performance of the proposed method, we have developed the following experimental system. We have modified the verifier of VINAS-P[13] such that it works as a partial order explorer for synthesis. It generates a timed state class graph which represents $\mathcal{T}_{S_i}(M)$. $\text{rsg}(\mathcal{T}_{S_i}(M), S_i)$ is then generated from it by a Perl script. Finally, the subcircuit is synthesized by using ATACS [12]. Note that in this experimental system, ATACS is used only for generating circuits from state graphs. Its state space exploration function is not used.

This section demonstrates the synthesis of the STARI circuit [14]. The STARI circuit is composed of a number of FIFO stages. The block diagram for a two-stage STARI circuit is shown in Fig.6(a). The purpose of this circuit is to absorb the clock skew between the sender and the receiver which are synchronous circuits with the same clock frequency. In the STARI circuit, each Boolean signal is represented by using the dual-rail code. Thus, $(x0t, x0f)$ takes $(1, 0)$ or $(0, 1)$ to represent the data “1” or “0”. $(0, 0)$ is used to separate each data. The sender sends each data at the rising edge of the clock and the separator at the falling edge. The receiver samples the data and produces the acknowledgment (active low) at the rising edge of the clock. At the falling edge, the separator is sampled, and the acknowledgment is reset. They are modeled by the time Petri nets in Fig.6(c) and (d). As shown in the figures, the output transitions in the sender and receiver have the earliest firing times 0 and latest firing times 1. Since those transitions are triggered by the clock edges, it models the clock skew from 0 to 1. Fig.6(e) shows the specification of a FIFO stage. It simply waits for the data from the preceding stage and the acknowledgment from the next stage, and propagates the data to the next stage as well as the acknowledgment to the preceding stage. Then, the resetting phase is followed. Our goal is to synthesize circuits for all stages.

Table I shows the CPU times (in seconds) for synthesizing STARI circuits with various number of stages by the proposed method and ATACS only without using the abstraction method described in [9]. They are measured on a UNIX workstation (1GHz, 2GB memory). It uses an improved POSET timing analysis algorithm [15]. Since the proposed system synthesizes only one subcircuit once, the CPU times shown in the table is the sum of CPU times for all runs. ATACS runs in nonmodular synthesis mode, so the CPU times are for generating all subcircuits.

Table II shows CPU times for the synthesis of each subcircuit of STARI 12. Two of the obtained subcircuits are shown in Fig.6(f). We believe that the difference in CPU times of VINAS-P comes from the difference of the concurrency between the signals of subcircuits. This table also shows the number of literals required for each stage of the STARI circuit. Note that although the STARI

TABLE I
PERFORMANCE COMPARISON.

n	Proposed	ATACS
8	1.0	5.6
9	1.0	10.7
10	1.2	25.2
11	2.0	95.5
12	15.3	1980.1
13	11.9	3509.6

TABLE II
CPU TIMES (SEC.) TO SYNTHESIZE STARI 12 SUBCIRCUITS.

stage ID	VINAS-P	Perl	ATACS	literals
1	0.33	0.13	0.04	4
2	0.37	0.16	0.03	4
3	0.36	0.13	0.04	4
4	0.95	0.18	0.05	4
5	2.02	0.18	0.03	10
6	4.81	0.24	0.04	10
7	1.47	0.17	0.03	10
8	1.03	0.20	0.03	10
9	0.64	0.11	0.04	10
10	0.32	0.11	0.05	8
11	0.29	0.13	0.06	8
12	0.29	0.11	0.06	8

circuit is a regular structure, the circuits for each stage do not need to be the same. In particular, the circuits at the beginning and the end of the FIFO circuit can be optimized using the available timing information. Those near the beginning appear as shown at the top of Fig.6(f). Note that since the *ack* signal is now not used by prior stages, the circuits for several of the early stages can be simply optimized to two wires. As expected, the circuits derived by the proposed method are identical to those found using flat synthesis.

VI. CONCLUSION

We have proposed an idea to apply partial order reduction to the modular synthesis of timed circuits. Each module from a hierarchical design is synthesized individually. Partial order reduction is utilized to reduce the state space explored for the other modules by only considering a single interleaving for concurrently enabled transitions. By avoiding the state explosion problem, the STARI circuit can be synthesized up to two orders of magnitude faster than the total order synthesis approach.

The proposed method can be improved, if we apply the POSET method to the analysis of the target subcircuit, and combine this work with the abstraction technique.

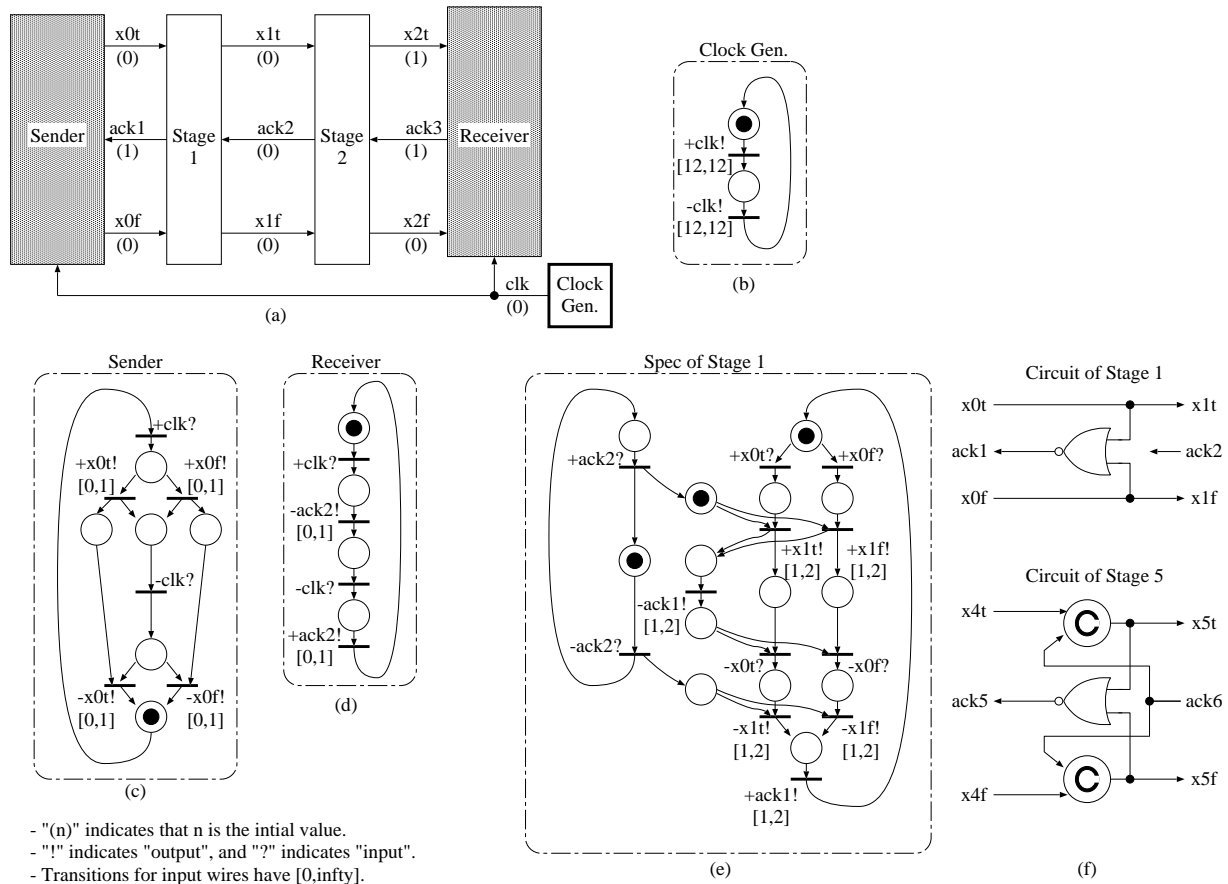


Fig. 6. Specification and environment of STARI circuit.

REFERENCES

- [1] P. Godefroid. Using partial orders to improve automatic verification methods. *Proc. of Computer Aided Verification Workshop*, 1990.
- [2] A. Valmari. A stubborn attack on state explosion. *Proc. of Workshop on Computer Aided Verification*, 1990.
- [3] S. Katz and D. Peled. Defining conditional independence using collapses. *Semantics for concurrency, BCS-FACS Workshop*, M. Kwiatkowska (ed.), Springer, 1990.
- [4] K. L. McMillan. *Symbolic model checking : An approach to the state explosion problem*. PhD thesis, Carnegie Mellon University, 1992.
- [5] T. Yoneda and H. Schlingloff. On model checking for Petri nets and a linear-time temporal logic. *IEICE Technical Report*, FTS92(1):1-8, 1992.
- [6] C. J. Myers, T. G. Rokicki, and T. H.-Y. Meng. POSET timing and its application to the synthesis and verification of gate-level timed circuits. *IEEE Transactions on Computer-Aided Design*, 18(6):769-786, June 1999.
- [7] Alex Semenov, Alexandre Yakovlev, Enric Pastor, Marco Pe na, Jordi Cortadella, and Luciano Lavagno. Partial order based approach to synthesis of speed-independent circuits. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 254-265. IEEE Computer Society Press, April 1997.
- [8] H. Zheng, E. Mercer, and C. Myers. Automatic abstraction for verification of timed circuits and systems. In *International Conference on Computer Aided Verification*. Springer-Verlag, 2001.
- [9] H. Zheng. *Modular Synthesis and Verification of Timed Circuits Using Automatic Abstraction*. PhD thesis, University of Utah, 2001.
- [10] T. Yoneda and H. Schlingloff. Efficient verification of parallel real-time systems. *Formal Method in System Design*, pages 187-215, 1997.
- [11] T. Yoneda and H. Ryu. Timed trace theoretic verification using partial order reduction. *Proc. of Fifth International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 108-121, 1999.
- [12] Chris J. Myers. *Computer-Aided Synthesis and Verification of Gate-Level Timed Circuits*. PhD thesis, Dept. of Elec. Eng., Stanford University, October 1995.
- [13] <http://yoneda-www.cs.titech.ac.jp/~yoneda/pub.html>.
- [14] Mark R. Greenstreet. Implementing a STARI chip. In *Proc. International Conf. Computer Design (ICCD)*, pages 38-43. IEEE Computer Society Press, 1995.
- [15] E. G. Mercer, C. J. Myers, and T. Yoneda. Improved POSET timing analysis in timed Petri nets. *Proc. of SASIMI'01*, 2001.