Automatic Abstraction for Verification of Timed Circuits and Systems*

Hao Zheng, Eric Mercer, and Chris Myers

University of Utah, Salt Lake City UT 84112, USA
{hao,eemercer,myers}@vlsigroup.elen.utah.edu
http://async.elen.utah.edu

Abstract. This paper presents a new approach for verification of asynchronous circuits by using automatic abstraction. It attacks the state explosion problem by avoiding the generation of a flat state space for the whole design. Instead, it breaks the design into blocks and conducts verification on each of them. Using this approach, the speed of verification improves dramatically.

1 Introduction

In order to continue to produce circuits of increasing speed, designers are considering aggressive circuit design styles such as self-resetting or delayed-reset domino circuits. These design styles can achieve a significant improvement in circuit speed as demonstrated by their use in a gigahertz research microprocessor (guTS) at IBM [11]. Designers are also considering asynchronous circuits due to their potential for higher performance and lower power as demonstrated by the RAPPID instruction length decoder designed at Intel [22]. This design was 3 times faster while using only half the power of the synchronous design. The correctness of these new *timed circuit* styles is highly dependent upon their timing, so extensive timing verification is necessary during the design process. Unfortunately, these new circuit styles cannot be efficiently and accurately verified using traditional static timing analysis methods. This lack of efficient analysis tools is one of the reasons for the lack of mainstream acceptance of these design styles.

The formal verification of timed circuits often requires state space exploration which can explode even for modest size examples. To reduce the complexity incurred by state exploration, abstraction is necessary. In [2,21], safe approximations of internal signal behavior are found to reduce the size of the state space, but these methods are still exponential in the number of memory elements. In VIS [6], non-determinism is used to abstract the behavior of some circuit signals, but it is often too conservative and can introduce unreachable states which may exhibit hazards. In [20], a model checker is proposed based on hierarchical reactive machines. By taking advantage of the hierarchy information, it only tracks active variables so that the state space is reduced and verification time is improved, but this approach is best suited for software which has a more sequential nature. In [16], an abstraction technique is proposed for validation coverage

* This research is supported by NSF CAREER award MIP-9625014, SRC contract 97-DJ-487 and 99-TJ-694, and a grant from Intel Corporation.

G. Berry, H. Comon, and A. Finkel (Eds.): CAV 2001, LNCS 2102, pp. 182–193, 2001.
© Springer-Verlag Berlin Heidelberg 2001

analysis and automatic test generation. It removes all datapath elements which do not affect the control flow and groups the equivalent transitions together resulting in a dramatic reduction in the state space. It is difficult, however, to distinguish the control from the datapath without help from the designers. In [13], an abstraction approach for the design of speed-independent asynchronous circuits from change diagrams is described. In this approach, each subcircuit is designed individually, and they are then recombined to produce the final circuit. This approach, however, does not address timing issues. In [10], a divide-and-conquer method is presented for the synthesis of asynchronous circuits. This method breaks up the state graph into a number of simpler subgraphs for each output, and each subgraph is solved individually. The results are then integrated together to construct the final solution. This method, however, requires a complete state graph to start with. An *assume-guarantee reasoning* strategy is shown [25]. In such cases, when verifying a component in a system, assumptions need to be made about the behavior of other components, and these assumptions are discharged when the correctness of other components is established. while our approach is similar to assume-guarantee reasoning, our approach does not require assumptions about the other components because their behavior is derived from the specifications using semantics-preserving abstraction. In [3], Belluomini describes the verification of domino circuits using `ATACS`. She shows that verifying flat circuits even of a moderate size can be very difficult, while the verification can be completed quickly using hand abstractions. However, these hand abstractions require an expert user and methods must be developed to check that the abstractions are a reliable model of the underlying behavior. This is the major motivation of this work.

Our approach begins with a high-level language, such as VHDL, that models a system hierarchically. The method then compiles each individual component into a timed Petri-net for verification. This paper proposes an abstraction technique applied to *timed Petri-nets*. This approach partitions the design into small blocks using specified structural information, and each block is verified separately. We have proven that under certain constraints if each block is verified to be correct, then the complete system is also correct. Our results show that taking advantage of the hierarchical information results in a substantial savings in verification time.

2 Timed Petri-Nets and Basic Trace Theory

Timed Petri-nets (TPNs) [19] are the graphical model to which our high-level specification is compiled. A one-safe TPN is modeled by the tuple $\langle P, T, F, M_0, \Delta \rangle$ where P is the set of places, T is the set of transitions, and $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation, $M_0 \subseteq P$ is the initial marking, and Δ is an assignment of timing constraints to places. There are three kinds of transitions: $s+$ changes signal s from 0 to 1, $s-$ changes s from 1 to 0, and \$ which is a *sequencing transition.* A *marking* is a subset of places. For a place $p \in P$, the *preset* of p (denoted $\bullet p$) is the set of transitions connected to p (i.e., $\bullet p = \{t \in T \mid (t, p) \in F\}$), and the *postset* of p (denoted $p\bullet$) is the set of transitions to which p is connected (i.e., $p\bullet = \{t \in T \mid (p, t) \in F\}$). Presets and postsets for transitions are similarly

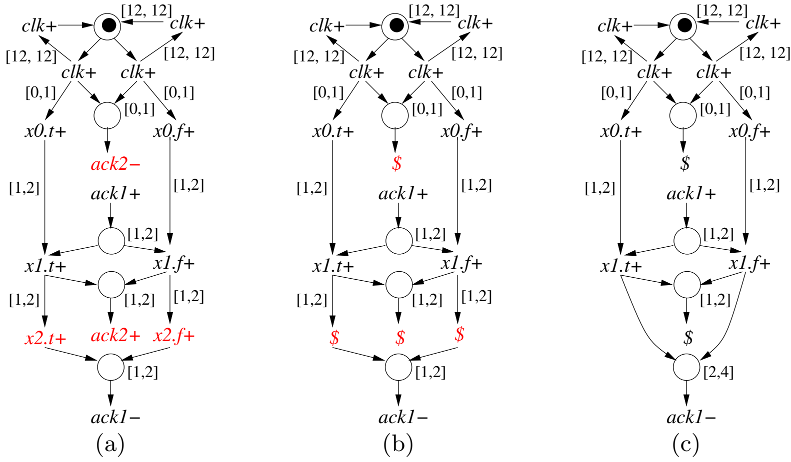


Fig. 1. Portion of the TPN for a STARI Circuit Composed of two FIFO Stages.

defined. A timing constraint consisting of a lower and upper bound is associated with each place in the TPN (i.e., $\Delta(p_i) = \langle l_i, u_i \rangle$). The lower bound is a non-negative integer while the upper bound is an integer greater than or equal to the lower bound or ∞. A benchmark for timed circuit design is the STARI communication circuit [9] which is used to communicate between two synchronous systems that are operating at the same clock frequency, but are out-of-phase due to clock skew. The STARI circuit is essentially a FIFO connecting the two systems. A portion of the TPN for a STARI circuit with 2 FIFO stages is shown in Figure 1(a). To simplify the diagram, places between transitions have been removed. A token indicates that the place is initially marked.

A transition t is enabled in a marking M if $\bullet t \subseteq M$. A `timer` is associated with each place $p \in M$. For each $p \in P$, $\texttt{timer}(p)$ is initialized to zero when p is put into the marking. All timers in a marking increase uniformly. Let $\texttt{lower}(p)$ and $\texttt{upper}(p)$ be the lower and upper bounds of the timing constraints of $p \in P$. For a $p \in M$, $\texttt{timer}(p)$ is *satisfied* if $\texttt{timer}(p) \geq \texttt{lower}(p)$; $\texttt{timer}(p)$ is *expired* if $\texttt{timer}(p) \geq \texttt{upper}(p)$. A transition t cannot occur until it is enabled in a marking and $\texttt{timer}(p)$ is satisfied for all $p \in \bullet t$. A transition t must fire before $\texttt{timer}(p)$ is expired for all $p \in \bullet t$. Firing a transition t changes the current marking M to a new marking $M' = (M - \bullet t) \cup t\bullet$, where $\texttt{timer}(p) = 0$ for all $p \in t\bullet$. The net is 1-safe if $(M - \bullet t) \cap t\bullet = \emptyset$.

The timing properties of a system are specified using a set of *constraint places*. Constraint places never actually enable a transition to fire. Instead, the constraint places are checked each time a transition fires in a marking. Failures caused by constraint places arise due to three conditions:

1. There exists a constraint place $p \in \bullet t$ such that $p \notin M$ when firing t.
2. $\texttt{timer}(p)$ is not satisfied for any constraint place $p \in \bullet t$ when firing t.
3. $\texttt{timer}(p)$ is expired for any constraint place $p \in \bullet t$ before firing t.

The dynamic behavior of a Petri net can be studied using reachability analysis. A marking M_n is said to be reachable from a marking M_0 if there exists a sequence of firings that changes M_0 to M_n. A *firing sequence* or *run* from a marking M_0 is defined as $\rho = M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} M_2 \xrightarrow{t_3} \ldots \xrightarrow{t_n} M_n$. A sequence of transitions generated by a firing sequence ρ is called a trace. In the above example, M_n is reachable from M_0 through a trace $t_1 t_2 \ldots t_n$. Let X be the set of all possible traces produced by a Petri net N. X is *prefix-closed* and always includes the empty trace ϵ.

The same concept can be extended to TPNs. A state S of a TPN is a pair $(M, \texttt{timer})$, where M is a marking and $\texttt{timer}$ is a function $p \rightarrow \mathbf{Q}^+$ for all $p \in M$. The initial state S_0 is $(M_0, \texttt{timer}_0)$, where $\texttt{timer}_0(p) = 0$ for all $p \in M_0$. In a state $S = (M, \texttt{timer})$, a transition t can fire if t is enabled in M and $\texttt{timer}(p)$ is satisfied for all $p \in \bullet t$. The new state $S' = (M', \texttt{timer}')$ is obtained from S by firing t. $M' = (M - \bullet t) \cup t\bullet$ and $\texttt{timer}'(p) = 0$ for all $p \in t\bullet$. A *timed firing sequence* or *timed run* in a TPN is defined as $\rho = S_0 \xrightarrow{t_1} S_1 \xrightarrow{t_2} S_2 \xrightarrow{t_3} \ldots \xrightarrow{t_n} S_n$, where S_0 is the initial state. S_{i+1} is obtained from S_i by passing some time until all rules in $\bullet t_{i+1}$ are satisfied and then firing t_{i+1}. Let $\texttt{time}_i(\rho)$ be the sum of time that has passed for the system to reach the state S_i from the initial state S_0 through the firing sequence ρ. It is true that $\texttt{time}_0(\rho) = 0$ and $\texttt{time}_{i+1}(\rho) = \texttt{time}_i(\rho) + \tau$ where $l \leq \tau \leq u$, $l = max(\{\texttt{lower}(r) | r \in M_i \text{ for } r \in \bullet t_{i+1}\})$, and $u = max(\{\texttt{upper}(r) | r \in M_i \text{ for } r \in \bullet t_{i+1}\})$. Thus, a run ρ produces a timed trace $(t_1, \texttt{time}_1(\rho))(t_2, \texttt{time}_2(\rho)) \cdots$. Let X be the set of all possible timed traces produced by a timed Petri net N. X is also *prefix-closed*.

Since the reachability analysis of a TPN can be uniquely determined by all its possible timed traces, the system behavior can also be described using *trace theory*. Trace theory has been applied to the verification of both speed-independent [8] and timed circuits [7,27]. A *timed trace*, x, is a sequence of events (i.e., $x = e_0 e_1 \ldots$). In trace theory, it is not necessary to distinguish the rising and falling transitions on the same signal, the signal name is used to represent both transitions on the same signal. Therefore, each *timed event* is of the form $e_i = (w_i, t_i)$ where w is a signal name in the TPN. t is a rational number indicating when a transition on a signal wire happens. A timed trace must satisfy the following two properties:

- *Monotonicity*: $t_i \leq t_{i+1}$ for all $i \geq 0$, and
- *Progress*: if x is infinite then for any time t there exists an i such that $t_i > t$.

The delete function, $\mathbf{del}(D)(x)$, removes all events of a trace $x = e_1 e_2 ...$ whose wire names are in a set D. More formally,

$$\mathbf{del}(D)(x) = \begin{cases} e_1 y & \text{if} \quad w_1 \notin D \\ y & \text{if} \quad w_1 \in D \end{cases} \tag{1}$$

where $y = \mathbf{del}(D)(e_2 e_3 ...)$ and $e_1 = (w_1, t_1)$. It is extended naturally to sets of traces. The inverse delete function, $\mathbf{del}^{-1}(D)(X)$, takes a set of wires, D, and a set of traces, X, and returns the set of traces which would be in X if all events with wire names in D are deleted (i.e., $\mathbf{del}^{-1}(D)(X) = \{x' \mid \mathbf{del}(D)(x') \in X\}$). Intuitively, if x is a trace not containing symbols from D, $\mathbf{del}^{-1}(D)(x)$ is the set

of all traces that can be generated by inserting events in D at any time into x. Some useful properties of these two functions are below:

$$\mathbf{del}(D)(X) = \emptyset \Leftrightarrow X = \emptyset \tag{2}$$

$$\mathbf{del}(D)(\mathbf{del}^{-1}(D')(X)) = \mathbf{del}^{-1}(D')(\mathbf{del}(D)(X)) \text{ when } D \cap D' = \emptyset \tag{3}$$

$$\mathbf{del}(D)(\mathbf{del}^{-1}(D)(X)) = X \tag{4}$$

$$\mathbf{del}(D)(X \cap X') \subseteq \mathbf{del}(D)(X) \cap \mathbf{del}(D)(X') \tag{5}$$

A *prefix-closed trace structure* T is a three-tuple $\langle I, O, P\rangle$. I is a set of input wires, and O is a set of output wires where $I \cap O = \emptyset$. $A = I \cup O$ is the *alphabet* of the structure. $P = S \cup F$ is the set of all possible traces of a system where S and F are the *success set* and the *failure set* of T, respectively. The trace structure T of a TPN N can be derived using state space exploration on N. A function $\mathbf{trace}(N)$ is defined to return a trace structure which has the same inputs and outputs as N. P of $\mathbf{trace}(N)$ is the set of all possible timed traces produced by N. The function $\mathbf{fail}(X)$ is defined to return the set of all traces in P that cause safety violations or timing constraint violations. Therefore, $F = \mathbf{fail}(P)$. For hierarchical verification to succeed, the definition of $\mathbf{fail}(X)$ must satisfy the following requirement:

$$\mathbf{fail}(X) \subseteq \mathbf{fail}(X') \quad \text{if} \quad X \subseteq X' \tag{6}$$

where X and X' are two sets of traces. This requirement states that for two sets of traces, correctness checking does not affect the relation of the two sets. S contains all successful traces of a system, and $S = P - F$. A trace structure must be *receptive*, meaning that $PI \subseteq P$. Intuitively, this means a circuit cannot prevent the environment from sending an input.

Composition ($\|$) combines two circuits into a single circuit. Composition of two trace structures $T = \langle I, O, S, F\rangle$ and $T' = \langle I', O', S', F'\rangle$ is defined when $O \cap O' = \emptyset$. To compose two trace structures, the alphabets of both trace structures must first be made the same by adding new inputs as necessary to each structure. Inverse delete is extended to trace structures for this step as follows:

$$\mathbf{del}^{-1}(D)(T) = \langle I \cup D, O, \mathbf{del}^{-1}(D)(S), \mathbf{del}^{-1}(D)(F)\rangle \tag{7}$$

This is defined only when $D \cap A = \emptyset$. After the two alphabets of the two structures are made to match, we need to find the traces that are consistent with the two structures. The intersection of these two trace structures is defined as follows:

$$T \cap T' = \langle I \cap I', O \cup O', S \cap S', (F \cap P') \cup (P \cap F')\rangle \tag{8}$$

This is defined only when $A = A'$ and $O \cap O' = \emptyset$. A success trace in the composite must be a success trace in both components. A failure trace in the composite is a possible trace that is a failure trace in either component. The possible traces for the composite is $P \cap P'$. Composition can now be defined:

$$T \parallel T' = \mathbf{del}^{-1}(A' - A)(T) \cap \mathbf{del}^{-1}(A - A')(T') \tag{9}$$

Another useful operation is **hide** which is used to make a set of wires, D, *internal* to the circuit. Given a trace structure T, $\mathbf{hide}(D)(T)$ is defined as follow:

$$\mathbf{hide}(D)(T) = \langle I, O - D, \mathbf{del}(D)(S), \mathbf{del}(D)(F)\rangle \tag{10}$$

A trace structure is *failure-free* if its failure set is empty. Given two trace structures, T and T', we say T *conforms* to T' (denoted $T \preceq T'$) if $I = I', O = O'$, and for *all* environments E, if $E \parallel T'$ is failure-free, so is $E \parallel T$. Intuitively, if a system using T' cannot fail, neither can a system using T.

Lemma 1 below gives a simple sufficient condition to determine conformance between two trace structures. The condition $F \subseteq F'$ assures that if the environment does not cause a failure in T', it does not cause a failure in T. The condition $P \subseteq P'$ assures that if T' does not cause a failure in the environment, T does not cause one. Lemma 2 shows that if T conforms to T', this conformance is maintained in any environment. Proofs of these lemmas can be found in [8].

Lemma 1. *$T \preceq T'$ if $I = I', O = O', F \subseteq F'$, and $P \subseteq P'$.*

Lemma 2. *If $T \preceq T'$ and T'' is any trace structure, then $T \parallel T'' \preceq T' \parallel T''$.*

3 Automatic Abstraction and Safe Transformations

Formal verification of timed systems is typically based on a complete exploration of the state space. The state space grows exponentially in the complexity of the design. This limits verification to small designs. In general, a large and complex design is organized as a number of components, each of which has a well-defined interface. To verify a timed system, an environment must be provided. The environment has two functions during verification. First, it defines and supplies the input behavior which the system must be able to process for correct operation. Second, the outputs of the system must not cause the environment to fail. Each component either connects to other components, the environment, or both. Since the complexity of each component is often much less than the whole system, it is desirable to verify each component individually, and integrate the results for all components when available to form the solution for the whole system. If a component is chosen for verification, the rest of the components and the system environment together form the environment in which the component operates. To verify a component, only the interface behavior of the environment is important to the component. Therefore, if the internal behavior of the environment is abstracted away while preserving its interface behavior, the environment can be simplified reducing the complexity of verification.

To apply abstraction to TPNs, first, all internal signals relative to a chosen component are identified and all transitions on them converted to sequencing transitions; second, these sequencing transitions and the related places are removed safely from the TPNs, when possible. Consider the TPN shown in Figure 1(a). If we are synthesizing only the first stage of the two stage FIFO, then the signals *ack2*, *x2.t*, and *x2.f* should be abstracted away. Transitions on these signals are changed to sequencing transitions as shown in Figure 1(b).

Next, transformations are applied to remove these sequencing transitions. Suzuhi and Murata [23,24] present a method of stepwise refinement of transitions and places into subnets. They show a sufficient condition that such subnets must satisfy which is dependent on the structure and initial marking of the net. The resulting net has the same liveness and safety properties as that of the

original net. This refinement process, however, has to be repeated every time the initial marking is changed. This makes automating the refinement difficult. Berthelot [5] presented several transformations that depend only on the structure of the net. In [18,12,17], several transformations for marked graphs are presented. These transformations reduce places and transitions in the graph while preserving liveness and safety. All these transformations, however, are only applied to untimed Petri nets.

We have developed several safe transformations for timed Petri nets. Safe transformations must obey two conditions. First, removal of a signal should never change the untimed semantics of the environment. Second, the timing information of the signal transitions produced by the environment must be preserved in a conservative fashion. To explain these two conditions more precisely, we use trace theory. Suppose N_E is the TPN describing the behavior of the environment, and T_E is its corresponding trace structure. The interface behavior of T_E is described by $\mathbf{del}(D)(P_E)$, where D is the set of signals internal to the environment, and P_E is the set of possible traces. The environment after abstraction and safe transformations is called the *abstracted environment*. In the abstracted environment, the internal signals, D, are removed from N_E to obtain the trace structure $T_A = \mathbf{trace}(\mathbf{abs}(D)(N_E))$. Function $\mathbf{abs}(D)(N_E)$ returns a TPN N'_E where the signals in D are abstracted away from N_E using safe transformations. Let X_1 and X_2 be the untimed trace sets produced by $\mathbf{abs}(D)(N_E)$ and N_E, respectively. To preserve the interface behavior, a safe transformation must satisfy that $X_1 = \mathbf{del}(D)(X_2)$ and $\mathbf{del}(D)(P_E) \subseteq P_A$, where D contains the internal signals of the environment to be removed and P_A is the possible trace set of T_A. Intuitively, this means a safe transformation should never remove any specified behavior, but it may add new behavior. In other words, the verification result might be a *false negative*, but never a *false positive*.

Figure 2 shows two simple transformations. Transformation 1 is used when a sequencing transition has a single or multiple places in its preset, and a single place in its postset. In transformation 2, the sequencing transition has a single place in its preset, and two or more places in its postset. While transformation 1 adds no extra behaviors, transformation 2 may create extra interleavings between b and c not seen before the transformation. For example, after the transformation, the system could generate a trace $(a, t_a)(c, t_a + l_1 + l_3)(e, t_a + u_2 + u_3)$, where t_a is when a fires. This trace is impossible in the system before the transformation.

The third transformation which involves a merge place is depicted in Figure 3. This transformation like the last one may add additional timing behavior. However, if $l_a = l_b$ and $u_a = u_b$ then it is an exact transformation. This transformation is applied to the TPN in Figure 1(b) to obtain the reduced one shown in Figure 1(c). Numerous other safe transformations have been developed and proven to be correct. Due to space limitations, these transformations and all proofs are omitted here, but can be found in [29].

In order to perform verification using TPNs, the possible traces P can be found using a timed state space exploration procedure such as the one described in [4]. After safe transformations, it is true that $\mathbf{del}(D)(P_E) \subseteq P_A$ where $T_A = \mathbf{trace}(\mathbf{abs}(D)(N_E))$ and D contains the internal signals to be removed. This indicates that the interface behavior of the environment after transformations is

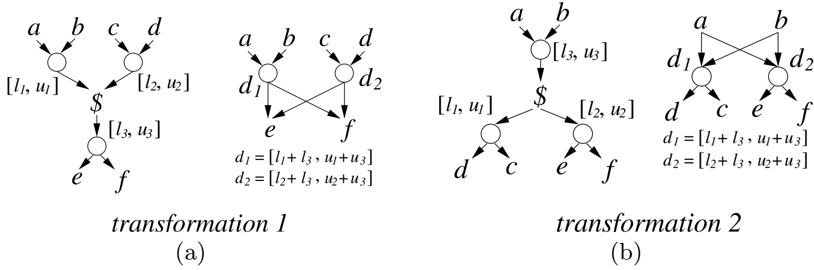


Fig. 2. Safe Transformation 1 and 2.

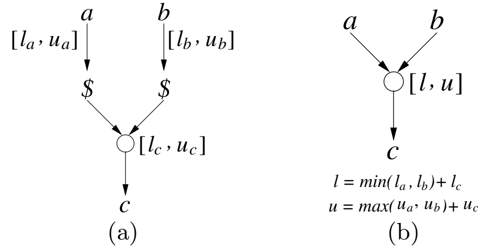


Fig. 3. Safe Transformation 3.

a superset of that before transformations. From Equation 6, we get the following:

$$\mathbf{fail}(\mathbf{del}(D)(P_E)) \subseteq \mathbf{fail}(P_A) \tag{11}$$

This means that the failure set of the abstracted environment is a superset of the failure set of the unabstracted environment with internal signals hidden. Based on the above discussion and Lemma 1, the following lemma can be proved easily.

Lemma 3. *Given a system described by a TPN, N_E, with a trace structure T_E, where D is the set of internal signals of the system. If the function $\mathbf{abs}(D)(N_E)$ uses only safe transformations, then $\mathbf{hide}(D)(T_E) \preceq \mathbf{trace}(\mathbf{abs}(D)(N_E))$.*

Hierarchical verification verifies each block in a system individually. If each block is verified to be failure-free with its abstracted environment, then we can prove that the entire system is failure-free. This idea is formalized in the following theorems. Given two modules $M_1 = \langle I_1, O_1, P_1 \rangle$ and $M_2 = \langle I_2, O_2, P_2 \rangle$, we would like to verify that their composition, $M_1 \parallel M_2$, is failure-free. In the following theorem, X_1 and X_2 are the internal signal sets of M_1 and M_2, respectively (i.e., $X_1 = O_1 - I_2$, $X_2 = O_2 - I_1$, and $X_1 \cap X_2 = \emptyset$).

Theorem 1. *Let X_1 and X_2 be the internal signal sets of M_1 and M_2, respectively. If $M_1 \parallel \mathbf{hide}(X_2)(M_2)$ is failure-free, and $\mathbf{hide}(X_1)(M_1) \parallel M_2$ is failure-free, then $M = M_1 \parallel M_2$ is failure-free.*

Proof: First, the failure set of $M_1 \parallel M_2$ is

$$(\mathbf{del}^{-1}(X_2)(\mathbf{fail}(P_1)) \cap \mathbf{del}^{-1}(X_1)(P_2)) \cup \\ (\mathbf{del}^{-1}(X_1)(\mathbf{fail}(P_2)) \cap \mathbf{del}^{-1}(X_2)(P_1)) \tag{12}$$

Suppose $M_1 \parallel \mathbf{hide}(X_2)(M_2)$ is failure-free. This means its failure set is empty.

$$(\mathbf{fail}(P_1) \cap \mathbf{del}^{-1}(X_1)(\mathbf{del}(X_2)(P_2))) \cup \\ (P_1 \cap \mathbf{del}^{-1}(X_1)(\mathbf{fail}(\mathbf{del}(X_2)(P_2)))) = \emptyset \tag{13}$$

$$\Rightarrow \ \mathbf{fail}(P_1) \cap \mathbf{del}^{-1}(X_1)(\mathbf{del}(X_2)(P_2)) = \emptyset \tag{14}$$

Using Equation 4, Equation 14 can be transformed to:

$$\mathbf{del}(X_2)(\mathbf{del}^{-1}(X_2)(\mathbf{fail}(P_1))) \cap \mathbf{del}^{-1}(X_1)(\mathbf{del}(X_2)(P_2)) = \emptyset \tag{15}$$

Using Equation 3, Equation 15 becomes:

$$\mathbf{del}(X_2)(\mathbf{del}^{-1}(X_2)(\mathbf{fail}(P_1))) \cap \mathbf{del}(X_2)(\mathbf{del}^{-1}(X_1)(P_2)) = \emptyset \tag{16}$$

From Equation 5, Equation 16 can be transformed to:

$$\mathbf{del}(X_2)(\mathbf{del}^{-1}(X_2)(\mathbf{fail}(P_1)) \cap \mathbf{del}^{-1}(X_1)(P_2)) = \emptyset \tag{17}$$

Finally, from Equation 2, we get the following result:

$$\mathbf{del}^{-1}(X_2)(\mathbf{fail}(P_1)) \cap \mathbf{del}^{-1}(X_1)(P_2) = \emptyset \tag{18}$$

Now, suppose $M_2 \parallel \mathbf{hide}(X_1)(M_1)$ is failure-free. In a similar manner, we derive:

$$\mathbf{del}^{-1}(X_1)(\mathbf{fail}(P_2)) \cap \mathbf{del}^{-1}(X_2)(P_1) = \emptyset \tag{19}$$

The union of Equation 18 and 19 is the failure set of $M_1 \parallel M_2$. Since both Equation 18 and 19 are empty, the failure set of $M_1 \parallel M_2$ is empty. ∎

Calculationof P is an exponential problem. Lemma 3 shows that $\mathbf{hide}(D)(T_E) \preceq \mathbf{trace}(\mathbf{abs}(D)(N_E))$. Therefore, from Lemma 2, $M_1 \parallel \mathbf{hide}(X_2)(M_2) \preceq M_1 \parallel \mathbf{trace}(\mathbf{abs}(X_2)(N_{M_2}))$ and $\mathbf{hide}(X_1)(M_1) \parallel M_2 \preceq \mathbf{trace}(\mathbf{abs}(X_1)(N_{M_1})) \parallel M_2$. Using the above conclusions, we show another very important theorem.

Theorem 2. *Let X_1 and X_2 be internal signal sets of M_1 and M_2, respectively. If $M_1 \parallel \mathbf{trace}(\mathbf{abs}(X_2)(M_2))$ is failure-free and $\mathbf{trace}(\mathbf{abs}(X_1)(M_1)) \parallel M_2$ is failure-free, then $M = M_1 \parallel M_2$ is failure-free.*

4 Results and Conclusions

We have incorporated our abstraction technique into our VHDL and HSE compiler [28] frontend to the `ATACS` tool. The charts in Figure 4 show the comparative runtimes for verification using POSET timing [4] with and without abstraction on two different FIFO circuits. Only the first few stages are shown as larger

FIFO's cannot be verified without abstraction for the first FIFO. ATACS completes 7 stages on the flat design; but with abstraction, it completes 100 stages in about 6.5 minutes. The second example is a multiple stage controller for a self-timed FIFO that is very timing dependent [15]. Without abstraction, only 4 stages can be analyzed. With abstraction, we can analyze 100 stages in 23 minutes.

The last example is the STARI communication circuit described in detail in [9]. The STARI circuit is used to communicate between two synchronous systems that are operating at the same clock frequency, but are out-of-phase due to clock skew. The STARI circuit is composed of a number of FIFO stages built from 2 C-elements and 1 NOR-gate per stage. There are two properties that need to be verified: (1) each data value output by the transmitter must be inserted into the FIFO before the next one is output and (2) a new data value must be output by the FIFO before each acknowledgment from the receiver [26]. To guarantee the second property, it is necessary to initialize the FIFO to be approximately half-full [9]. In [26], the authors state that COSPAN which uses a region technique for timing verification [1] ran out of memory attempting to verify a 3 stage gate-level version of STARI on a machine with 1 GB of memory. This paper goes on to describe an abstract model developed by hand for STARI for which they could verify 8 stages in 92.4 MB of memory and 1.67 hours. A flat gate-level design for 10 stages can be verified in 124 MB and 20 minutes using POSET timing [4]. Our automated abstraction method verifies a 14 stage STARI with a maximum memory usage of 23 MB of memory for a single stage in about 5 minutes. Figure 5 shows the comparative runtimes for verification with and without abstraction on STARI using Bap, an enhanced version of the POSET timing analysis algorithm [14]. As shown in the chart, Bap can verify STARI for up to 12 stages with a memory usage of 277 MB. In the first few stages, the runtime for verification with abstraction is larger because abstraction itself takes time. When the complexity of the design grows, the runtime for flat verification grows much faster.

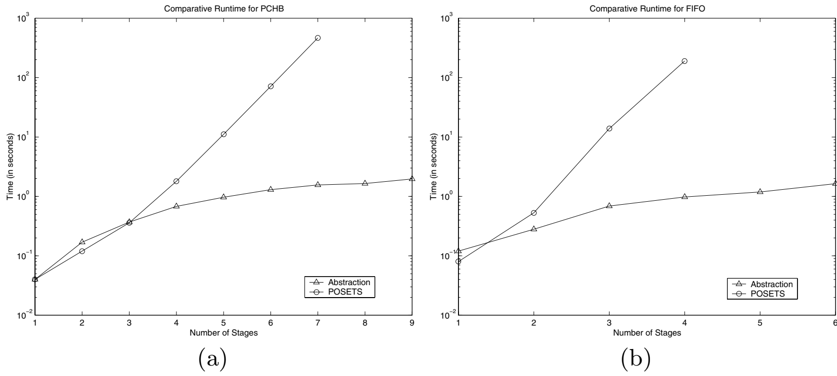
Since abstraction runtime grows polynomially in the size of the specification, the total runtime with abstraction grows in an approximately polynomial manner. This is substantially better than the exponential growth in the analysis of flat designs. We have also found that verification with abstraction is not only several orders of magnitude faster than that for flat designs, but also successful on several orders of magnitude more complex designs.

Acknowledgments

We would like to thank Wendy Belluomini of IBM and Tomohiro Yoneda of the Tokoyo Institute of Technology, Japan for their helpful comments.

References

1. R. Alur and R. P. Kurshan. Timing analysis in cospan. In *Hybrid Systems III*. Springer-Verlag, 1996.



(a) (b)

Fig. 4. Runtimes for PCHB and FIFO Example.

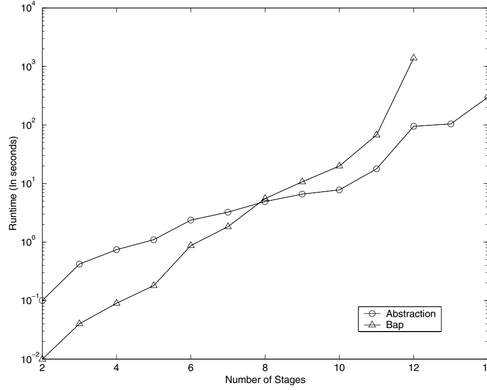


Fig. 5. Runtimes for Stari.

2. Peter A. Beerel, Teresa H.-Y. Meng, and Jerry Burch. Efficient verification of determinate speed-independent circuits. In *Proc. International Conf. Computer-Aided Design (ICCAD)*, pages 261–267. IEEE Computer Society Press, November 1993.
3. W. Belluomini, C. J. Myers, and H. P. Hofstee. Verification of delayed-reset domino circuits using ATACS. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 3–12, April 1999.
4. W. Belluomini and C.J. Myers. Verification of timed systems using posets. In *International Conference on Computer Aided Verification*. Springer-Verlag, 1998.
5. G. Berthelot. Checking properties of nets using transformations. In *Lecture Notes in Computer Science, 222*, pages 19–40, 1986.
6. R. K. Brayton. Vis: A system for verification and synthesis. In *Proc. International Conf. Computer-Aided Design (ICCAD)*, pages 428–432, 1996.
7. J. R. Burch. *Trace Algebra for Automatic Verification of Real-Time Concurrent Systems*. PhD thesis, Carnegie Mellon University, 1992.
8. David L. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits*. ACM Distinguished Dissertations. MIT Press, 1989.

9. M. R. Greenstreet. Stari: Skew tolerant communication. unpublished manuscript, 1997.
10. J. Gu and R. Puri. Asynchronous circuit synthesis with boolean satisfiability. In *IEEE Trans. CAD, Vol. 14, No. 8*, pages 961–973, 1995.
11. H. P. Hofstee, S. H. Dhong, D. Meltzer, K. J. Nowka, J. A. Silberman, J. L. Burns, S. D. Posluszny, and O. Takahashi. Designing for a gigahertz. *IEEE MICRO*, May-June 1998.
12. R. Johnsonbaugh and T. Murata. Additional methods for reduction and expansion of marked graphs. In *IEEE TCAS, vol. CAS-28, no. 1*, pages 1009–1014, 1981.
13. Michael Kishinevsky, Alex Kondratyev, Alexander Taubin, and Victor Varshavsky. *Concurrent Hardware: The Theory and Practice of Self-Timed Design*. Series in Parallel Computing. John Wiley & Sons, 1994.
14. E. Mercer, C. Myers, and Tomohiro Yoneda. Improved poset timing analysis in timed petri nets. Technical report, University of Utah, 2001. `http://www.async.utah.edu`.
15. Charles E. Molnar, Ian W. Jones, Bill Coates, and Jon Lexau. A FIFO ring oscillator performance experiment. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 279–289. IEEE Computer Society Press, April 1997.
16. D. Moundanos, J. Abraham, and Y. Hoskote. Abstraction techniques for validation coverage analysis and test generation. *IEEE TC*, 47(1):2–14, 1998.
17. T. Murata. Petri nets: Properties, analysis, and applications. In *Proceedings of the IEEE 77(4)*, pages 541–580, 1989.
18. T. Murata and J. Y. Koh. Reduction and expansion of lived and safe marked graphs. In *IEEE TCAS, vol. CAS-27, no. 10*, pages 68–70, 1980.
19. C. Ramchandani. *Analysis of Asynchronous Concurrent Systems by Timed Petri Nets*. PhD thesis, MIT, Feb. 1974.
20. R. Alur R.Grosu and M. McDougall. Efficient reachability analysis of hierarchical reactive machines. In *12th International Conference on Computer-Aided Verification, LNCS 1855*, pages 280–295, 2000.
21. Oriol Roig. *Formal Verification and Testing of Asynchronous Circuits*. PhD thesis, Univsitat Politècnia de Catalunya, May 1997.
22. Shai Rotem, Ken Stevens, Ran Ginosar, Peter Beerel, Chris Myers, Kenneth Yun, Rakefet Kol, Charles Dike, Marly Roncken, and Boris Agapiev. RAPPID: An asynchronous instruction length decoder. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 60–70, April 1999.
23. I. Suzuki and T. Murata. *Stepwise refinements for transitions and places*. New York: Springer-Verlag, 1982.
24. I. Suzuki and T. Murata. A method for stepwise refinements and abstractions of petri nets. In *Journal Of Computer System Science, 27(1)*, pages 51–76, 1983.
25. S. Tasiran, R. Alur, R. Kurshan, and R. Brayton. Verifying abstractions of timed systems. In *LNCS*, volume 1119, pages 546–562. Springer-Verlag, 1996.
26. S. Tasiran and R. K. Brayton. Stari: A case study in compositional and heirarchical timing verification. In *Proc. International Conference on Computer Aided Verification*, 1997.
27. Tomohiro Yoneda and Hiroshi Ryu. Timed trace theoretic verification using partial order reduction. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 108–121, April 1999.
28. Hao Zheng. Specification and compilation of timed systems. Master's thesis, University of Utah, 1998.
29. Hao Zheng. *Automatic Abstraction for Synthesis and Verification of Timed Systems*. PhD thesis, University of Utah, 2001.