Verification of Bounded Delay Asynchronous Circuits with Timed Traces

Tomohiro Yoneda, Bin Zhou[1], and Bernd–Holger Schlingloff[2]

[1] Tokyo Institute of Technology {yoneda,zhou}@cs.titech.ac.jp
[2] Universität Bremen, TZI-BISS hs@tzi.org

Abstract. In this paper, we extend the verification method based on the failure semantics of process algebra and the resulting trace theory by Dill et al. for bounded delay asynchronous circuits. We define a timed conformance relation between trace structures which allows to express both safety and responsiveness properties. In our approach, bounded delay circuits as well as their real-time properties are modelled by time Petri nets. We give an explicit state-exploration algorithm to determine whether an implementation conforms to a specification. Since for IO-conflict free specifications the conformance relation is transitive, this algorithm can be used for hierarchical verification of large asynchronous circuits. We describe the implementation of our method and give some experimental results which demonstrate its efficiency.

Keywords: Trace Theory, Time Petri Nets, Failure Analysis, Conformance Checking, State Space Exploration, Asynchronous Circuits, Hardware Verification, Delay Analysis, Real Time Systems, Computer Aided Verification

1 Introduction

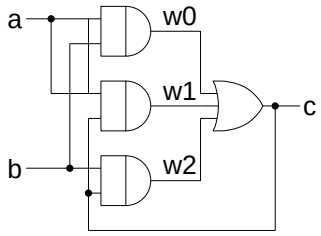
One of the main problems in the design of wafer-scale integrated circuits is the distribution of the global clock signal. Difficulties which arise in the design of large synchronous circuits are clock skews, clock delay estimation in layout design, etc. Therefore, *asynchronous processors* without a global clock are of increasing interest. However, asynchronous circuits are difficult to construct since the timing analysis often is very complex. Because of this reason, asynchronous circuits are usually modelled with a *speed independent model*, where the gate delays are unbounded, or are bounded by an unknown constant. Most of the research on design, synthesis, and verification of asynchronous circuits has been done under this model. Although the speed independent model is quite powerful, the possibility of unbounded delay can force the designer to add additional complexity to the circuit. For example, Muller's C element [MB 59], defined by the truth table in Fig. 1(a), is implemented by the circuit of Fig. 1(b).

This implementation, however, is not correct under the speed independent model. Assuming that each gate can have an unbounded delay, there exists a

a	b	c
0	0	0
0	1	P
1	0	P
1	1	1

P : previous value

(a)



(b)

Fig. 1. Muller's C element: truth table and gate-level implementation.

signal transition sequence in which the output illegally goes down before both inputs go down (suppose all wires initially have the value 0) :

$$a\uparrow\ b\uparrow\ w_0\uparrow\ c\uparrow\ b\downarrow\ w_0\downarrow\ c\downarrow .$$

The reason for this alleged fault is an extremely large delay of the gate with output w_1. With any well-processed VLSI, such a large delay should be impossible. In actual designs, the given circuit can be safely used to implement a C element. Thus, the speed independent model sometimes is not appropriate. In this paper we use a *bounded delay model* to model asynchronous circuits, where with each gate a lower and upper bound for the delay is associated.

In [Dil 88], an efficient verification method for speed independent circuits was proposed, which is based on *trace theory*. The primary advantage of this method is the possibility of hierarchical verification, which greatly reduces the complexity of the verification procedure. However, this method is only suited for verifying safety properties.

In this paper we adapt Dill's verification method to the bounded delay model. First, we show how trace theory can be extended to handle timed traces as well as certain timing requirements. We then describe time Petri nets as an appropriate model for asynchronous bounded delay circuits. Subsequently, we derive an algorithm to check whether an implementation, consisting of a set of modules, meets its specification. Finally, we give some experimental results and concluding remarks.

2 Timed Trace Theory

Let us briefly describe verification based on trace theory. In this method, the specification of a circuit is given as a *trace automaton*, i.e. a finite automaton over an input alphabet $\mathcal{I}$ and output alphabet $\mathcal{O}$. The implementation, which is supposed to be a set of modules, is given as a set of trace automata, each one representing the behavior of its related module. Then, special *composition* and *hiding* operations on trace automata are defined. The implementation *conforms*

to the specification, if they agree on the input and output alphabets, respectively, and the implementation can be safely substituted for the specification in every context. This means, that the implementation causes a failure in an environment only if the specification also causes a failure in that environment.

A *failure* of a module in an environment is an output of the module which is not accepted by the environment, or an output of the environment which is not accepted by the module. By this definition, conformance can be expanded to the following requirements: the implementation should be able to handle every input that the specification can handle, and it never produces an output unless the specification can produce it. This in turn can be checked by considering the mirror of the specification, where all inputs are outputs and vice versa. The implementation conforms to the specification iff the result of hiding all internal signal transitions in the implementation and composing it with the mirror of the specification is failure-free.

The verification approach proposed here is the timed version of this method, where time Petri nets and timed traces are used instead of automata and traces. The extension to real-time makes it also possible to verify certain timing properties.

In the rest of this section, we define timed traces and their related notions, and the conformance relation between specification and implementation.

Let $\mathcal{W}$ be a set of wires, and let $\mathbf{Q}$ denote the set of nonnegative rational numbers. For any $w \in \mathcal{W}$ and $t \in \mathbf{Q}$, the tuple (w, t) is called an *event*. Intuitively, (w, t) represents the change of the value of wire w at time t.

Definition 1. *A* (timed) trace *x over $\mathcal{W}$ is a finite or infinite sequence of events $x \triangleq x_1 x_2 \cdots$, where $x_i \triangleq (w_i, t_i)$, such that the following properties are satisfied:*

- *Monotonicity: for all $0 < i < |x|$, $t_i \leq t_{i+1}$.*
- *Progress: if x is infinite, then for every $t \in \mathbf{Q}$ there exists an index i such that $t_i > t$.*

In this definition, $|x|$ denotes the length of trace x. If $|x| = 0$, then x is the *empty trace* ε. For any finite trace x, trace y, and event e, the result of *appending* e or y to x is denoted by $x \circ e$ or $x \circ y$, respectively. x is a *prefix* of y if $y = x$ or $y = x \circ z$ for some trace z. The *projection* of a trace $x \triangleq x_1 \circ x_2 \circ \cdots$ over $\mathcal{W}$ onto another alphabet $\mathcal{W}'$ can be defined as usual:

$$project(x, \mathcal{W}') \triangleq \begin{cases} \varepsilon, & \text{if } x = \varepsilon \\ x_1 \circ y, & \text{if } x_1 = (w_1, t_1),\ w_1 \in \mathcal{W}' \\ y & \text{else,} \qquad \text{where } y \triangleq project(x_2 \circ x_3 \circ \cdots, \mathcal{W}') \end{cases}$$

Definition 2. *A* module *or* canonical trace structure *is a tuple $M \triangleq (\mathcal{I}, \mathcal{O}, \mathcal{T})$, where $\mathcal{I}$ is a set of* input wires, *$\mathcal{O}$ is a set of* output wires *($\mathcal{I} \cap \mathcal{O} = \emptyset$), and $\mathcal{T}$ is a set of traces over $\mathcal{W} \triangleq \mathcal{I} \cup \mathcal{O}$.*

The traces of a module can be regarded as the set of all maximal execution sequences of some transition system. However, trace structures are insensitive to

nondeterminism; they can not distinguish between $a \circ (b+c)$ and $(a \circ b)+(a \circ c)$. In timed systems, usually the set of traces will be an infinite (or even uncountable) set of infinite sequences.

Now we consider the composition of several modules. Assume we are given a set $\mathcal{M} \triangleq \{M_1, \cdots, M_n\}$ of modules, where $M_k \triangleq (\mathcal{I}_k, \mathcal{O}_k, \mathcal{T}_k)$, $\mathcal{W}_k \triangleq \mathcal{I}_k \cup \mathcal{O}_k$, and $\mathcal{O}_j \cap \mathcal{O}_k = \emptyset$. That is, each wire is either an input, output, or both; in the latter case we say the wire is *internal*. Any wire can be an output of at most one module, and input of arbitrary many modules. Intuitively, modules are composed by soldering wires with the same name together. Output wires of one module are connected to input wires of other modules. However, in some cases this connection of wires may cause failures in the composed module.

If $M = (\mathcal{I}, \mathcal{O}, \mathcal{T})$, then M *without* w $(M \setminus w)$ is the module $(\mathcal{I}', \mathcal{O}', \mathcal{T}')$, where $\mathcal{I}' = \mathcal{I} - \{w\}$, $\mathcal{O}' = \mathcal{O} - \{w\}$ and $\mathcal{T}' = \mathit{project}(\mathcal{T}, \mathcal{I}' \cup \mathcal{O}')$. Module M *allows* trace x $(M \models x)$ if there exists some trace y such that x is a prefix of y and $\mathit{project}(y, \mathcal{W}) \in \mathcal{T}$. Furthermore, for $\mathcal{M} \triangleq \{M_1, ..., M_n\}$, we say that $\mathcal{M} \models x$ if $M_k \models x$ for all $k \leq n$.

Definition 3. *A* safety failure *of* $\mathcal{M}$ *is any nonempty finite trace* $x \triangleq y \circ (w, t)$, *where* $w \in \mathcal{O}_k$ *for some* $k \leq n$, *such that* $\mathcal{M} \setminus w \models x$, *and* $M_k \models x$, *but* $\mathcal{M} \not\models x$.

Intuitively, a safety failure occurs if any module M_k tries to send an output, but some other module cannot receive this as internal input. $\mathcal{M}$ is *safety failure free*, if no safety failure can occur, i.e., if every output which may be produced by some module can be accepted by all other modules at the same time. Whenever a module can change the value on one of its output wires, all modules which have this wire connected as internal input must be able to process the signal immediately.

Definition 4. *A* timing failure *of* $\mathcal{M}$ *is any nonempty finite trace* $x \triangleq y \circ (w, t)$, *where* $w \in \mathcal{I}_k$ *for some* $k \leq n$, *such that* $\mathcal{M} \setminus w \models x$, *and* $M_k \models x$, *but there is no* $x' \triangleq y \circ (w', t')$, *where* $w' \in \mathcal{I}_k$, *and* $\mathcal{M} \models x'$.

Intuitively, a timing failure occurs if some module M_k expects an internal input from some other module which is not provided in time. $\mathcal{M}$ is timing failure free if whenever a module requests a signal on one of its internal input wires, there exists a module which can produce some signal as output within the required time interval. For any set $\mathcal{M} \triangleq \{M_1, \cdots, M_n\}$ of modules, $\mathit{failure}(\mathcal{M})$ is the set of all safety and timing failures of $\mathcal{M}$. $\mathcal{M}$ is *failure-free* if $\mathit{failure}(\mathcal{M}) = \emptyset$.

Next, we define a conformance relation between a system consisting of a set of modules and a specification given as a single module. Consider a set $\mathcal{M}_C \triangleq \{M_1, \cdots, M_n\}$ of modules, where $M_k \triangleq (\mathcal{I}_k, \mathcal{O}_k, \mathcal{T}_k)$, and a module $M_S \triangleq (\mathcal{I}_S, \mathcal{O}_S, \mathcal{T}_S)$ such that $\mathcal{I}_S \triangleq \bigcup \mathcal{I}_k - \bigcup \mathcal{O}_k$ and $\mathcal{O}_S \subseteq \bigcup \mathcal{O}_k$. Module M_S can be thought of as an abstract specification of the concrete circuit M_C: all external inputs of the circuit M_C appear as inputs of the specification M_S, and some (but not necessarily all) outputs of the circuit M_C are visible in the specification M_S.

Definition 5. $\mathcal{M}_C$ conforms to M_S, *if for any module* $M_E \triangleq (\mathcal{O}_S, \mathcal{I}_S, \mathcal{T}_E)$, *whenever* $\{M_S, M_E\}$ *is failure-free, also* $\mathcal{M}_C \cup \{M_E\}$ *is failure-free.*

In other words, the circuit $\mathcal{M}_C$ may have a failure in the environment M_E only if the specification M_S allows a failure in the same context. This conformance relation is reflexive and transitive, but not symmetric: The circuit may be failure-free even in contexts in which the specification fails.

A module M is called *I/O-conflict free*, if for any trace x, and for all events $e_i \triangleq (w_i, \tau_i)$ and $e_o \triangleq (w_o, \tau_o)$ with $w_i \in \mathcal{I}$ and $w_o \in \mathcal{O}$ it holds that $M \models x \circ e_i$ and $M \models x \circ e_o$ implies $M \models x \circ e_i \circ e_o$ and $M \models x \circ e_o \circ e_i$. Since conflicts between inputs and outputs often indicate hazardous situations, specifications usually do not contain such conflicts. Thus, henceforth we assume that all specifications are I/O-conflict free.

Definition 6. *The* mirror module M^m *of a module* $M \triangleq (\mathcal{I}, \mathcal{O}, \mathcal{T})$ *is the module* $M^m \triangleq (\mathcal{O}, \mathcal{I}, \mathcal{T})$*; that is, each input wire in* M^m *is an output wire of* M *and vice versa.*

For any module M, the set $\{M, M^m\}$ is failure-free. Moreover, the following *hierarchy lemma* holds:

Lemma 1. *Consider three modules* M_1, M_2 *and* M_3 *such that* $\mathcal{I}_1 = \mathcal{I}_2 = \mathcal{I}_3$ *and* $\mathcal{O}_1 \supseteq \mathcal{O}_2 = \mathcal{O}_3$*. If* $\{M_1, M_2^m\}$ *is failure-free and* $\{M_2, M_3^m\}$ *is failure-free, then* $\{M_1, M_3^m\}$ *is failure-free.*

Proof. Assume that both $\{M_1, M_2^m\}$ and $\{M_2, M_3^m\}$ are failure-free, and let $x \triangleq y \circ (w, t)$. We have to show that $\{M_1, M_3^m\}$ is failure-free. The following cases have to be considered.

1. First, assume that $w \in \mathcal{O}_1$, $M_1 \models x$ and $M_3^m \setminus w \models x$, and show that $M_3^m \models x$. If $w \notin \mathcal{W}_2$, then from $\mathcal{W}_3 \subseteq \mathcal{W}_2$, we have $w \notin \mathcal{W}_3$. Thus, from $M_3^m \setminus w \models x$, we have $M_3^m \models x$. If $w \in \mathcal{W}_2$, then w cannot be from $\mathcal{I}_2 = \mathcal{O}_2^m$, since in this case it would be impossible to compose M_1 with M_2^m. Hence w must be in $\mathcal{O}_2$. Assume for contradiction that $M_2 \setminus w \not\models x$. Then there must be some $x' = y' \circ (w', t')$ such that x' is an initial part of x, $M_2 \setminus w' \models y'$ and $M_2 \setminus w \not\models x'$. Since $M_1 \models x$ and $M_3 \setminus w \models x$, it follows that $M_1 \models x'$ and $M_3 \setminus w \models x'$. Since $M_2 \setminus w \not\models x'$, we must have $w' \in \mathcal{W}_2$. If $w' \in \mathcal{O}_2$, then $w' \in \mathcal{O}_1$. Since $\{M_1, M_2^m\}$ is safety failure-free, $M_2 \setminus w \models x'$, which is a contradiction. Similarly, if $w' \in \mathcal{I}_2 = \mathcal{I}_3 = \mathcal{O}_3^m$, then since $\{M_2, M_3^m\}$ is safety failure-free, a contradiction arises. Thus, $M_2 \setminus w \models x$. Since $\{M_1, M_2^m\}$ is safety failure-free, $M_2^m \models x$, thus $M_2 \models x$. Since $\{M_2, M_3^m\}$ is safety failure-free, $w \in \mathcal{O}_2$ and $M_3^m \setminus w \models x$ (hypothesis), we have $M_3^m \models x$.
2. The second case is symmetric to the first case: assume that $w \in \mathcal{O}_3^m = \mathcal{I}_3$, $M_3^m \models x$ and $M_1 \setminus w \models x$, and show that $M_1 \models x$. To be able to compose M_1 with M_3^m, the set $\mathcal{O}_1 \cap \mathcal{O}_3^m = \mathcal{I}_3$ must be empty. If $w \notin \mathcal{I}_1$, then $M_1 = M_1 \setminus w$ and there is nothing to show. If $w \in \mathcal{I}_1$, then $\mathcal{I}_1 \subseteq \mathcal{I}_2$ gives $w \in \mathcal{I}_2 = \mathcal{O}_2^m$. Similar to the previous case, $M_2 \setminus w \models x$. Since $\{M_2, M_3^m\}$ is safety failure-free, we can infer that $M_2 \models x$. Since $\{M_1, M_2^m\}$ is safety failure-free, it follows that $M_1 \models x$.

3. Next, assume that $w \in \mathcal{I}_1$, $M_1 \models x$ and $M_3^m \setminus w \models x$, and show that $\{M_1, M_3^m\} \models y \circ (w', t')$ for some $w' \in \mathcal{I}_1$. Since $\{M_1, M_2^m\}$ does not have timing failures, there is some (w_1, t_1) with $w_1 \in \mathcal{I}_1$ such that $\{M_1, M_2^m\} \models y \circ (w_1, t_1)$. Since $\mathcal{I}_1 \subseteq \mathcal{I}_2$ and $\{M_2, M_3^m\}$ does not have timing failures, there is some (w_2, t_2) with $w_2 \in \mathcal{I}_2$ such that $\{M_2, M_3^m\} \models y \circ (w_2, t_2)$. Since $w_2 \in \mathcal{I}_2 = \mathcal{O}_2^m$ and $\{M_1, M_2^m\}$ does not have safety failures, $M_1 \models y \circ (w_2, t_2)$.
4. Finally, assume that $w \in \mathcal{I}_3^m = \mathcal{O}_3$, $M_3^m \models x$ and $M_1 \setminus w \models x$, and show that $\{M_1, M_3^m\} \models y \circ (w', t')$ for some $w' \in \mathcal{O}_3$. Since $\{M_2, M_3^m\}$ is timing failure-free, there is some (w_1, t_1) such that $w_1 \in \mathcal{O}_3$ and $\{M_2, M_3^m\} \models y \circ (w_1, t_1)$. Since $\mathcal{O}_3 \subseteq \mathcal{O}_2 = \mathcal{I}_2^m$ and $\{M_1, M_2^m\}$ is timing failure-free, there is some (w_2, t_2) such that $w_2 \in \mathcal{O}_2$ and $\{M_1, M_2^m\} \models y \circ (w_2, t_2)$. Since $\{M_2, M_3^m\}$ is safety failure-free and $\mathcal{O}_2 \subseteq \mathcal{O}_3$, we have $\{M_1, M_3^m\} \models y \circ (w_2, t_2)$ as desired. □

This lemma can be extended to deal with sets of modules instead of a single modules. From the hierarchy lemma, the following *mirror theorem* can be obtained. It gives a similar characterization of conformance as in [Dil 88]:

Theorem 1. *$\mathcal{M}_C$ conforms to M_S iff $\mathcal{M}_C \cup \{M_S^m\}$ is failure-free.*

Proof. Assume that $\mathcal{M}_C \cup \{M_S^m\}$ has a failure. Then for the environment $M_E = M_S^m$ we have that $\{M_S, M_E\}$ is failure-free, but $\mathcal{M}_C \cup \{M_E\}$ is not failure-free, i.e., $\mathcal{M}_C$ does not conform to M_S.
In the other direction, we have to show that failure-freeness of $\mathcal{M}_C \cup \{M_S^m\}$ implies that $\mathcal{M}_C$ conforms to M_S. Since M_S is a specification for the circuit $\mathcal{M}_C$, $\mathcal{I}_S = \bigcup_k \mathcal{I}_k - \bigcup_k \mathcal{O}_k$ and $\mathcal{O}_S \subseteq \bigcup_k \mathcal{O}_k$. If $\mathcal{M}_C \cup \{M_S^m\}$ is failure-free, then the hierarchy lemma asserts that for any module M_E such that $\mathcal{W}_E = \mathcal{W}_S$ and $\{M_S, M_E\}$ is failure-free, $\mathcal{M}_C \cup \{M_E\}$ must also be failure-free. Thus, $\mathcal{M}_C$ conforms to M_S. □

To get an intuitive understanding of the conformance relation, consider the case of a single module $M_C \triangleq (\mathcal{I}_C, \mathcal{O}_C, \mathcal{T}_C)$ conforming to $M_S \triangleq (\mathcal{I}_S, \mathcal{O}_S, \mathcal{T}_S)$. This amounts to $\mathcal{I}_S = \mathcal{I}_C$, $\mathcal{O}_S \subseteq \mathcal{O}_C$, and for all traces x such that $\{M_C, M_S\} \models x$, and all events $i \triangleq (w_i, t_i)$, $w_i \in \mathcal{I}_S$, and $o \triangleq (w_o, t_o)$, $w_o \in \mathcal{O}_S$, the following holds:

- If $M_S \models x \circ i$, then $M_C \models x \circ i$,
- if $M_C \models x \circ o$, then $M_S \models x \circ o$,
- if $M_S \models x \circ o$, then there exists an $o' \triangleq (w_o', t_i')$, $w_o' \in \mathcal{O}_S$ such that $\{M_S, M_C\} \models x \circ o'$, and
- if $M_C \models x \circ i$, then there exists a $i' \triangleq (w_i', t_i')$, $w_i' \in \mathcal{I}_S$ such that $\{M_S, M_C\} \models x \circ i'$.

The first and second condition state that $\{M_C, M_S^M\}$ is safety failure-free: every input allowed by M_S is allowed by M_C, and every output allowed by M_C is allowed by M_S. The third condition reflects the definition of timing-failure: as long as M_S^m expects an input, that is, M_S requires an output, M_C should produce some output in time. The fourth condition is similar. If M_C is constructed as an implementation for the specification M_S, then this can be read as:

- The implementation can handle every input that the specification can handle,
- the implementation never produces an output unless the specification produces it,
- if the specification requires an output, the implementation produces it in time, and
- the implementation never expects an input unless the specification expects the input.

Therefore, our definition of the conformance relation includes not only safety properties, but also a certain timing property. In the case of bounded delay asynchronous circuits the absence of timing failure amounts to in-time-responsiveness, which is an important issue for verification. For example, consider the specification of an *or*-gate, where input a or b lead to output c within a certain time. Suppose that this specification is implemented erroneously by an *and*-gate. Then, after sending a to this circuit, it can not produce the output c. However, since the specification requires such an output, this situation leads to a timing failure.

Note that we do not actually compose the modules constituting the implementation. Therefore, in our approach it is not necessary to eliminate so-called autofailures, which arise from internal communication errors in a composed module. Also we do not have an explicit hiding operation: Failures resulting from the effect of hiding variables are transparent to the specification and will also be detected during the verification procedure. However, if we consider only safety-failures in untimed systems, then our notion of conformance is equivalent to the one in [Dil 88].

3 Analysis of Time Petri Nets

In the general setting of the previous section, there was absolutely no restriction posed on the set of traces of a module. To be able to give concrete algorithms, however, this set should at least be recursive, i.e., generated by some kind of automaton. In this section, we consider trace sets generated by one-safe *time Petri nets* [MF 76]. In contrast to timed Petri nets or stochastical Petri nets, which are used in *simulation* for the optimization of processes [TSS 98], time Petri nets have been applied successfully in the *verification* of hard real-time constraints.

One-safe Petri nets can be seen as a subclass of finite automata, where different parallel activities can be modelled by multiple tokens. Therefore, a Petri net model can be much more succinct than the corresponding automaton. Similarly, time Petri nets can be regarded as a subclass of timed automata [AD 92]. Compared with timed automata, the expressive power of time nets with respect to certain timing properties is restricted. This restriction, however, simplifies the analysis: we can check the conformance relation by a simple state space generation algorithm, traversing every state only once in a depth first search manner.

Definition 7. *A* time Petri net N *is a six-tuple,* $N \triangleq (P, T, F, Eft, Lft, \mu_0)$, *where*

- $P \triangleq \{p_1, p_2, \cdots, p_m\}$ *is a finite nonempty set of* places;
- $T \triangleq \{\tau_1, \tau_2, \cdots, \tau_n\}$ *is a finite set of* transitions $(P \cap T = \emptyset)$;
- $F \subseteq (P \times T) \cup (T \times P)$ *is the* flow relation;
- $Eft : T \to \mathbf{Q}$, $Lft : T \to \mathbf{Q} \cup \{\infty\}$ *are functions for the* earliest *and* latest firing times *of transitions, satisfying* $Eft(\tau) \leq Lft(\tau)$ *for all* $\tau \in T$;
- $\mu_0 \subseteq P$ *is the* initial marking *of the net.*

For any transition τ, $\bullet\tau \triangleq \{p \in P \mid (p, \tau) \in F\}$ and $\tau\bullet \triangleq \{p \in P \mid (\tau, p) \in F\}$ denote the *preset* and the *postset* of τ, respectively.

In the following we will restrict ourselves to one-safe Petri nets, where each place can contain at most one token. Therefore, a *marking* μ of N is defined to be any subset of P. A transition is *enabled* in a marking μ if $\bullet\tau \subseteq \mu$ (all its input places have tokens in μ); otherwise, it is *disabled*. Let $enabled(\mu)$ be the set of transitions enabled in μ.

A *state* σ of a time Petri net is a pair $(\mu, clock)$, where μ is a marking and *clock* is a function $T \to \mathbf{Q}$. The *initial state* σ_0 is $(\mu_0, clock_0)$, where $clock_0(\tau) = 0$ for all $\tau \in T$.

The states of time Petri nets change, if time passes or if a transition fires. In state $\sigma \triangleq (\mu, clock)$, time $t \in \mathbf{Q}$ *can pass*, if for all $\tau \in enabled(\mu)$, $clock(\tau) + t \leq Lft(\tau)$. In this case, state $\sigma' \triangleq (\mu', clock')$ is *obtained by passing* t from σ, if

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

In state $\sigma \triangleq (\mu, clock)$, transition $\tau_f \in T$ *can fire*, if $\tau_f \in enabled(\mu)$, and $clock(\tau_f) \geq Eft(\tau_f)$. In this case, state $\sigma' \triangleq (\mu', clock')$ is *obtained by firing* τ_f from σ, if

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

Intuitively, this can be interpreted as follows: Passing time t does not change the marking, but advances all clock values. Firing a transition τ_f consumes no time, but updates μ and *clock* such that the clock values associated with newly enabled transitions (i.e. transitions which are enabled in μ' but not in $\mu - \bullet\tau_f$) are reset to 0. Clock values of other transitions (i.e. transitions not affected by τ_f) are left unchanged.

In contrast to untimed Petri nets, not all enabled transitions may be firable in a given state; certain firing sequences which can occur without timing may not be possible in the time Petri net. A *run* $\rho \triangleq \sigma_0 \stackrel{\tau_1}{\to} \sigma_1 \stackrel{\tau_3}{\to} \sigma_2 \stackrel{\tau_3}{\to} \cdots$ of N is a finite or infinite sequence of states and transitions such that σ_0 is the initial state, and σ_{i+1} is obtained from σ_i by passing time and then firing transition τ_{i+1}. We write $\sigma_i(\rho)$ for the i-th state of ρ, and similarly $\mu_i(\rho)$ and $clock_i(\rho)$, and

omit the argument (ρ) whenever appropriate. A run is *maximal*, if it is infinite or in its last state there is no enabled transition. The *behavior* $B(N)$ of N is the set of all maximal runs of N.

Given any run ρ and $i \geq 0$, we define $time_i(\rho)$ to be the sum of all times t passed between $\sigma_0(\rho)$ and $\sigma_i(\rho)$; that is, $time_0(\rho) \triangleq 0$ and $time_{i+1}(\rho) \triangleq time_i(\rho) + clock_{i+1}(\tau) - clock_i(\tau)$ for some τ which is not newly enabled in μ_{i+1}. A state σ is *reachable* if there exists a finite run whose last state is σ.

Definition 8. *A time Petri net is* one-safe, *if for every state* $\sigma \triangleq (\mu, clock)$ *obtained by passing time from any reachable state* σ', *and for every transition* τ *which can fire in* σ, $\tau\bullet \cap \mu = \emptyset$.

The restriction to one-safe nets simplifies the verification algorithm.

In order to satisfy the progress condition, we assume that time certainly passes in any cyclic behavior of N. For example, this requirement is satisfied if the sum of earliest firing times of transitions forming any loop in N is positive. In the sequel, a *net* will always be a one-safe time Petri net satisfying the above restriction.

Let *wire* be a function from a set of transitions to a set of wires. Every maximal run $\rho \triangleq \sigma_0 \stackrel{\tau_1}{\rightarrow} \sigma_1 \stackrel{\tau_2}{\rightarrow} \cdots$ of a net N *generates* the timed trace $((wire(\tau_1), time_1(\rho)), (wire(\tau_2), time_2(\rho)), \cdots)$. We also say that a net N *represents* the module consisting of all traces generated by maximal runs of N.

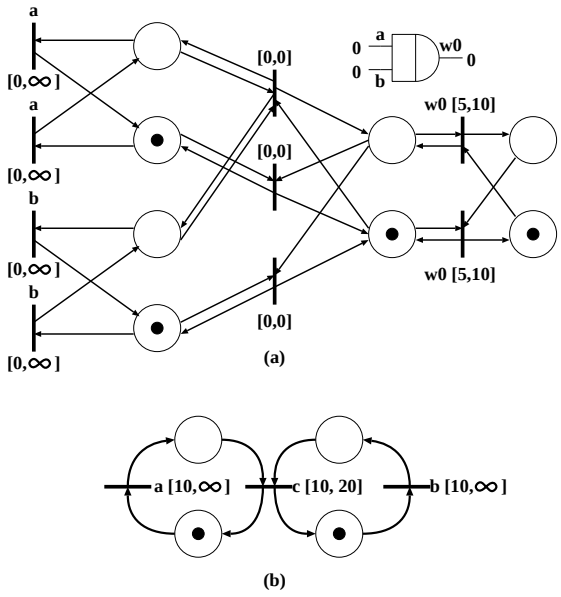


Fig. 2. Nets specifying AND gate and C element

Bounded delay asynchronous circuits can be easily described by nets. For example, an *and*-gate which has inputs a, b and an output w_0 with gate delay [5,10] can be represented by the net shown in Fig. 2(a). In this modelling, we do not distinguish between the change of a wire from 0 to 1 and from 1 to 0. An *or*-gate can be represented similarly. Even though it would be possible to give a more detailled description of gates (e.g., transistor level behavior), for most verification purposes the given net is an adequate representation.

The composition of several gates in a circuit can be described by simply putting together all nets representing single gates. Assuming that all wires in the circuit have unique names, for each transition the corresponding wire can be assigned. Then, the disjoint union of all these nets represents the complete circuit. Thus, the implementation of Muller's C element shown in Fig. 1(b) can be represented by a collection of nets which are similar to the one in Fig. 2(a).

This implementation works correctly under the following assumptions:

1. if an input changes, then the same input never changes again before an output changes, and
2. no input changes before some constant time passes after the change of the output.

The net shown in Fig. 2(b) specifies the behavior of a C-element with these assumptions. Verification consists in showing that the gate-level representation conforms to this specification. This is done by exploring the reachable states of the composed net.

We now describe an algorithm to generate these reachable states of time Petri nets. Since for time Petri nets the time domain consists of rational (not real) numbers, the state space can be finitely represented by sets of systems of inequalities. Basically, we use a system of inequalities to represent a number of different clock functions of time Petri nets. By an *inequality* we mean any string of the form "$x - y \sim c$", where x and y are from a designated set of variables, $c \in \mathbf{Q}$ and $\sim$ is a relation symbol from $\{\leq, \geq\}$. If I is a set of inequalities, then $var(I)$ denotes the set of variables that I contains; we say that I is a set of inequalities *over* $var(I)$. Let I be a set of inequalities over $\{x_1, x_2, \cdots, x_m\}$. A *feasible vector* for I is a tuple $(c_1, c_2, \cdots, c_m)$ of constants $c_i \in \mathbf{Q}$, such that every inequality obtained by replacing every x_i by c_i $(1 \leq i \leq m)$ in any inequality from I holds in the theory of rational numbers. The *solution set* of I is the set of feasible vectors for I. A set of inequalities is *consistent* if its solution set is nonempty. Two sets of inequalities are *isomorphic*, if they have the same solution set.

If the net $N \triangleq (P, T, F, Eft, Lft, \mu^0)$ represents the module $M \triangleq (\mathcal{I}, \mathcal{O}, \mathcal{T})$, we denote this by $M = (\mathcal{I}, \mathcal{O}, N, wire)$. An *abstract state* of the net is a pair (μ, I), where $\mu \subseteq P$ and I is a set of inequalities. Each abstract state denotes an equivalence class of reachable states of the net, namely all states for which the clock values form a feasible vector in the solution set of I. The initial abstract state of N is (μ^0, I_0), where $I_0 \triangleq \{\text{"}Eft(\tau) \leq \underline{\tau} - v \leq Lft(\tau)\text{"} \mid \tau \in enabled(\mu^0)\}$. Here, $\underline{\tau}$ in I_0 is a variable to represent the next firing time of the transition τ. The variable v indicates the initial time point.

The next step is to compute the set of abstract successor states σ' of an abstract state σ of N. To this end we need the notion of *deletion* of a set U of variables from a set I of inequalities. For every such I and U there exists an (up to isomorphism) unique set $I' \triangleq delete(I, U)$ of inequalities over $var(I) - U$, such that the solution set of I' is equal to the solution set of I, projected on $var(I) - U$. For example, if $I \triangleq \{$"$y - x \geq 2$", "$y - x \leq 7$", "$y - z < 3$", "$z - y \leq 11$"$\}$, then $delete(I, \{y\}) = \{$"$x - z < 1$", "$z - x \leq 18$"$\}$. I' can be computed incrementally by a shortest path algorithm in time $O(|var(I)|^2)$[Rok 93,Shi 94].

Let $\sigma \triangleq (\mu, I)$ be an abstract state of N, and $\tau_f \in enabled(\mu)$. Then, $first(\mu, \tau_f) \triangleq \{$"$\underline{\tau} - \underline{\tau_f} \geq 0$"$\mid \tau \in enabled(\mu)\}$ is a set of inequalities describing that τ_f is the first transition which fires in μ. $firable(\sigma) \triangleq \{\tau_f \mid \tau_f \in enabled(\mu), I \cup first(\mu, \tau_f)$ is consistent$\}$ is the set of transitions that can fire earlier than all other transitions in the given marking.

- τ_f is a transition in $firable(\sigma)$.
- μ' is the marking of N obtained by firing transition τ_f.
  That is, $\mu' \triangleq (\mu - {}^\bullet\tau_f) \cup \tau_f{}^\bullet$.
- R is a set of newly enabled transitions obtained by the firing of τ_f. That is,
  $R \triangleq enabled(\mu') - enabled(\mu - {}^\bullet\tau_f)$.
- $J \triangleq \{$"$\underline{\tau} - \underline{\tau_{out}} \geq Eft(\tau)$"$\mid \tau \in R\} \cup \{$"$\underline{\tau} - \underline{\tau_{out}} \leq Lft(\tau)$" $\mid \tau \in R\}$.
- $J' \triangleq I \cup first(\mu, \tau_{out}) \cup J$.
- $D \triangleq \{\underline{\tau} \mid \tau$ made some transition τ' enabled, and τ' is still enabled in $\mu'\}$.
- $I' \triangleq delete(J_3, \{\underline{\tau} \mid \tau \notin enabled(\hat{\mu'})\} - D)$

Intuitively, J, J', D and I' can be read as follows: J relates the variables of newly enabled transitions to the variable of the fired transition τ_{out}. J' is the union of I, J, and a set of inequalities representing that τ_{out} fires earlier than others. Transitions related to variables in D are currently parents of enabled transitions in μ', and these variables are necessary to check the coverability between the firing domains of transitions. Finally, in I' the variables of disabled transitions except for those in D are deleted. We write $\sigma \xrightarrow{\tau_f} \sigma'$ if $\sigma' \triangleq (\mu', I')$ is a successor of the abstract state $\sigma \triangleq (\mu, I)$ with respect to τ_f.

We now describe how conformance can be checked, using this successor relation between abstract states. We consider a set $\{M_0, M_1, \cdots, M_n\}$ of modules, where $M_i = (\mathcal{I}_i, \mathcal{O}_i, N_i, wire_i)$, $N_i \triangleq (P_i, T_i, Eft_i, Lft_i, \mu_i^0)$, and assume that for $i \neq j$, $P_i \cap P_j = T_i \cap T_j = P_i \cap T_j = \emptyset$. Some module in the set is a mirror of a specification, and input transitions and output transitions must not be in conflict in the module. If there is no confusion, we use the notation $wire$ instead of $wire_i$, and $\tau \in M_i$, when $\tau \in T_i$. Let $m(\tau)$ be the module number of τ, i.e., $m(\tau) = i$, if $\tau \in M_i$. Transition τ is called an *output transition* if $wire_{m(\tau)}(\tau) \in \mathcal{O}_{m(\tau)}$, and an *input transition* if $wire_{m(\tau)}(\tau) \in \mathcal{I}_{m(\tau)}$. If $\sigma_i \triangleq (\mu_i, I_i)$, $i \leq n$, are abstract states of the nets N_i, and K is a set of inequalities, we say that $s \triangleq (\sigma_0, \cdots, \sigma_n, K)$ is an abstract state of the module set $\{M_0, M_1, \cdots, M_n\}$.

The initial abstract state is $s_0 \triangleq (\sigma_0^0, \cdots, \sigma_n^0, \emptyset)$. We extend the definitions of $enabled(\mu)$ and $firable(\sigma)$ with respect to $s \triangleq (\sigma_0, \cdots, \sigma_n, K)$ as follows.

$$enabled(s) \triangleq \{\tau \mid \tau \in enabled(\mu_{m(\tau)})\}, \text{ and}$$

$$globally\_firable(s) \triangleq \{\tau \mid \tau \in enabled(s), \quad first(s,\tau) \cup \bigcup_{i=0}^{n} I_i \cup K \text{ is consistent}\},$$

where $first(s,\tau) \triangleq \{\text{“}\tau - \tau' \leq 0\text{”} \mid \tau' \in enabled(s)\}$. Furthermore, for an output transition τ_O such that $\tau_O \in globally\_firable(s)$,

$$sync\_trans(\tau_O, s) \triangleq \{\tau \mid wire(\tau) = wire(\tau_O), \tau \in globally\_firable(s)\}.$$

When $\{M_0, M_1, \cdots, M_n\}$ is at $s \triangleq (\sigma_0, \cdots, \sigma_n, K)$, it moves to $s' \triangleq (\sigma_0', \cdots, \sigma_n', K')$ with respect to $\tau_O \in globally\_firable(s)$ by firing all transitions in $sync\_trans(\tau_O, s)$.

- for $1 \leq i \leq n$
  - if $\tau \in sync\_trans(\tau_O, s) \cap T_i$, then $\sigma_i \xrightarrow{\tau} \sigma_i'$, and
  - if $sync\_trans(\tau_O, s) \cap T_i = \emptyset$, then $\sigma_i' = \sigma_i$.
- $K' \triangleq K \cup \{\text{“}\underline{\tau} = \underline{\tau'}\text{”} \mid \tau, \tau' \in sync\_trans(\tau_O, s)\}$.

Let $s \xrightarrow{\tau_O} s'$ denote this state transition relation of the module set. For any transition τ and abstract state σ, the variable $parent(\tau, \sigma)$ indicates which transition enabled τ. Formally, if $\sigma \triangleq (\mu, I)$, $\sigma' \triangleq (\mu', I')$, $\sigma \xrightarrow{\tau} \sigma'$, and $\tau' \in enabled(\sigma')$, then

$$parent(\tau', \sigma') \triangleq \begin{cases} \underline{\tau}, & \text{if } \tau' \in enabled(\mu') - enabled(\mu - \bullet\tau) \\ parent(\tau', \sigma), & \text{otherwise.} \end{cases}$$

For a set I of inequalities, let $earlier(x, y, I)$ be the predicate expressing that $solution(\{\text{“}x > y\text{”}\} \cup I) = \emptyset$, i.e., $earlier(x, y, I)$ holds iff $x \leq y$ for every solution vector of I. We write $earlier(x, y, \sigma_i)$ for $earlier(x, y, I_i)$, where $\sigma_i \triangleq (\mu_i, I_i)$, and $earlier(x, y, s)$ for $earlier(x, y, \bigcup_{i=0}^{n} I_i \cup K)$, where $s \triangleq (\sigma_0, \cdots, \sigma_n, K)$. Let $\tau \in M_i$, $\sigma_i \triangleq (\mu_i, I_i)$, and $\tau \in enabled(s)$.

- $earliest\_firing\_time(s, \tau) \triangleq parent(\tau, \sigma_i) + Eft(\tau)$, and
- $latest\_firing\_time(s, \tau) \triangleq parent(\tau, \sigma_i) + Lft(\tau)$.

A state $s \triangleq (\sigma_0, \cdots, \sigma_n, K)$ is called *safe*, if for every output transition τ_O such that $\tau_O \in globally\_firable(s)$, and for every module $M_j (0 \leq j \leq n)$ such that $wire(\tau_O) \in \mathcal{I}_j$, there exists an input transition τ_I such that $wire(\tau_I) = wire(\tau_O), \tau_I \in enabled(s)$, $earlier(earliest\_firing\_time(s, \tau_I), \tau_O, s)$ holds, and either

1. $earlier(\tau_O, latest\_firing\_time(s, \tau_I), s)$, or
2. for some output transition τ such that $\tau \in enabled(s)$, $earlier(\tau, latest\_firing\_time(s, \tau_I), s)$.

A state $s \triangleq (\sigma_0, \cdots, \sigma_n, K)$ is called *live*, if for every input transition τ_I such that $\tau_I \in globally\_firable(s)$, there exists an output transition τ (of an arbitrary module) such that $\tau \in globally\_firable(s)$.

Let modules $M_1, \cdots, M_n$ be represented by nets $N_1, \cdots, N_n$. A safety failure corresponds to a non-safe state in the reachable state space, and a timing failure occurs if a state can be reached which is not live. In other words, $failure(M_1, M_2, \cdots, M_n)$ is empty, iff every state which is reachable from the initial state of $\langle N_1, \cdots, N_n \rangle$ is both safe and live. Therefore, the verification of conformance between modules can be done by traversing the state space of $\langle N_1, \cdots, N_n \rangle$ and checking if non-safe or non-live states are reachable.

Furthermore, it is possible to replace an abstract description of a module by a more concrete implementation. If $\{M_1, \cdots, M_{k-1}, M_k, M_{k+1}, \cdots, M_n\}$ conforms to M_S, $\{M_{k_1}, ..., M_{k_m}\}$ conforms to M_k, and $(\bigcup_{j=1}^{m} \mathcal{W}_{k_j} - \mathcal{W}_k) \cap \bigcup_{j=1}^{n} \mathcal{W}_j = \emptyset$, then $\{M_1, \cdots, M_{k-1}, M_{k_1}, ..., M_{k_m}, M_{k+1}, \cdots, M_n\}$ conforms to M_S. The set of wires in a specification usually is much smaller than the set of wires in the implementation. Thus, the total computation cost to determine whether $\{M_1, \cdots, M_{k-1}, M_k, M_{k+1}, \cdots, M_n\}$ conforms to M_S and $\{M_{k_1}, \cdots, M_{k_m}\}$ conforms to M_k is significantly smaller than the computation of whether $\{M_1, \cdots, M_{k-1}, M_{k_1}, \cdots, M_{k_m}, M_{k+1}, \cdots, M_n\}$ conforms to M_S. This is the primary advantage of hierarchical verification.

4 Experimental Results

We have implemented the algorithm shown in the previous section on a UNIX workstation in C++. In this section, we present some experimental verification results.

First, our verifier shows that the implementation in Fig. 1(b) is correct with respect to the specification in Fig. 2(b) after traversing 51 states, which takes about one second on a 17 MIPS workstation.

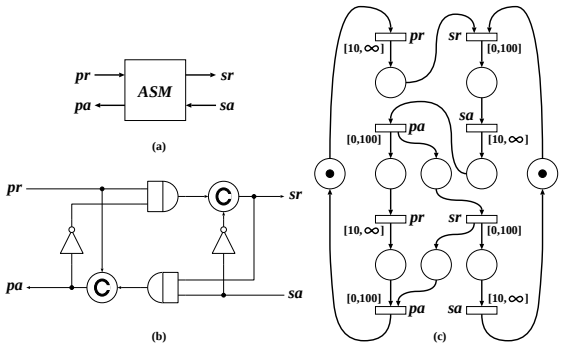


Fig. 3. An automatic sweeping module, gate level implementation, and specification

The second example is a control circuit of the request-acknowledgement handshake mechanism for asynchronous circuits. This circuit called an automatic sweeping module (ASM, for short) has two inputs (a primary request *pr*, a secondary acknowledgement *sa*) and two outputs (a primary acknowledgement *pa*, a secondary request *sr*) (Fig. 3(a)). It has the following functionality:

1. When the primary request goes high with the secondary acknowledgement low, ASM sets the secondary request.
2. When the secondary acknowledgement becomes high, ASM resets the secondary request with setting the primary acknowledgement.
3. When the primary request becomes low, ASM resets the primary acknowledgement.

This functionality with almost the same assumptions as for the C-element is specified with a net as shown in Fig. 3(c). On the other hand, Fig. 3(b) was proposed as the gate level implementation of ASM. We assume that each gate has a delay [5,10].

Our verifier shows that this implementation is correct with respect to the specification in Fig. 3(c). In Table 1, the column *flat* shows the size of the nets, the number of states, and CPU times needed for this verification when C elements are expanded by using their gate level implementations shown in Fig. 1. The column *hierarchical* shows the results of the hierarchical verification. That is, the specification net shown in Fig. 2(b) is used for the verification of ASM. In this case, the total verification time is the sum of the verification times for both ASM and C-element. These results show the advantage of the hierarchical verification as well.

Table 1. Results of verification

	flat			*hierarchical*		
	size†	states	CPU time (s)	size†	states	CPU time (s)
C-element	–	–	–	p:30, t:34	51	1.3
ASM	p:78, t:90	391	81.8	p:34, t:34	58	1.2
Total	p:78, t:90	391	81.8	p:64, t:68	109	2.5

† : "p:" and "t:" represent the numbers of places and transitions, respectively.

5 Conclusion

In this paper, we have extended the trace theoretic verification method for speed-independent asynchronous circuits to handle bounded delay asynchronous circuits. Our method is based on timed traces, and can check (timed) safety properties as well as responsiveness properties. It also inherits from the original method the possibility of hierarchical verification.

We use time Petri nets to describe both specification and implementation. Time Petri nets are a natural extension of ordinary Petri nets, which are widely

used in conventional verification methods. In this formalism, both (timed) properties and bounded delay asynchronous circuits can be described. We have developed a decision algorithm to check whether an implementation is correct with respect to its specification. It is based on state space traversal of a set of time Petri nets, and checking if any failure states are reachable.

First experimental results show that hierarchical verification works extremely well. Nevertheless, the increase of the number of modules can have a bad influence on the verification time. In the future we want to apply partial order analysis techniques [YY 96,YS 97,BM 98] to our method. This could help to further reduce the average complexity of the verification.

Acknowledgments
We would like to thank D. Dill for giving us the program code of his verifier, and Ichiki Honma for his help in implementing a first version of our algorithm.

References

[AD 92] R. Alur and D. Dill: *Automata for Modelling Real-time Systems*; LNCS 600: Real time: Theory in Practice, pp.45–73, (1992).

[BM 98] W. Belluomini and C. Myers: *Verification of Timed Systems Using POSETS* Proc. Int. Conf. on Computer Aided Verification (CAV), Univ. of British Columbia, Vancouver, Canada (1998)

[BD 91] 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).

[Dil 88] D. Dill: *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits*; PH.D. Thesis, MIT press, (1988).

[GTM 96] J.D. Garside, S. Temple, and R. Mehra: *The AMULET2e Cache System*; Proc. of the Second International Symposium of ASYNC, pp. 208—217 (1996).

[JM 87] F. Jahanian and A. Mok: *A Graph-Theoretic Approach for Timing Analysis and its Implementation*; IEEE Trans. Comput. C-36(8):961–975 (1987).

[MB 59] D. Muller and W. Bartke: *A Theory of Asynchronous Circuits*; Theory of Switching, Harvard University Press, Masachusetts (1959).

[MF 76] P. Merlin and D. Faber: *Recoverability of Communication Protocols*; IEEE Trans. on Communication, COM-24(9), (1976).

[Rok 93] T. Rokicki: *Representing and Modeling Digital Circuits*; Ph.D Dissertation, Stanford University, (1993).

[Shi 94] W. Shigetaka: *On the Acceleration of Formal Verification Methods*; Thesis, Tokyo Institute of Technology (1994).

[TSS 98] L. Twele, B.-H. Schlingloff, H. Szczerbicka: *Performability Analysis of an Avionics-Interface*; IEEE Symp. on Man, Machine and Cybernetics; San Diego, (1998)

[YS 97] T. Yoneda, B-H. Schlingloff: *Efficient Verification of Parallel Real-Time Systems*; Journal of Formal Methods in System Design 11-2, pp. 187-215, (1997).

[YY 96] T. Yoneda, T. Yoshikawa: *Using Partial Orders for Trace Theoretic Verification of Asynchronous Circuits*; Proc. of the Second International Symposium of ASYNC (1996).