Distributed Computing (1986) 1:197–204

DISTRIBUTED COMPUTING

A formal model for defining and classifying delay-insensitive circuits and systems

Jan Tijmen Udding
Institute for Biomedical Computing, Washington University, St. Louis, MO 63110, USA

Jan Tijmen Udding *received the B.S. and M.S. degrees in mathematics, and the Ph.D. degree in computer science from the Eindhoven University of Technology, Eindhoven, The Netherlands, in 1975, 1980, and 1984, respectively. Currently he is an Assistant Professor with the Department of Computer Science at Washington University, St. Louis, Missouri, and an Associate Professor with the Department of Computer Science at the Eindhoven University of Technology. His research interests are mathematical aspects of VLSI, concurrency, program derivation and correctness, and functional programming.*

Introduction

VLSI technology appears to be a powerful medium to realize highly concurrent computations. The fact that we can now fabricate systems that are more complex and more parallel makes high demands, however, upon our ability to design reliable systems. Although almost all circuits designed nowadays are clocked, we discuss and advocate the use of asynchronous circuits, because both bridling the complexity and improving the reliability of VLSI circuits seems to call for an asynchronous design approach, as is argued in the next two paragraphs. Moreover, asynchronous circuits are potentially faster.

The reason why asynchronous circuits may reduce the design complexity is that parameters determining a circuit's behavior do not scale in the same way, when the size of that circuit is scaled down. Seitz (1979) argues that scaling down a circuit's size by dividing all dimensions by a factor α results in a transition switching delay that is α times shorter. The propagation time for an electrical signal between two points on a wire, however, is the same as the propagation time for an electrical signal between the two corresponding points in the scaled circuit. Therefore, in VLSI circuits the relationship between wire delay and switching delay becomes such that delays in connecting wires might not be neglected anymore.

Bridling the design complexity is usually achieved by a modular approach. This will work only if the design of these modules can be independent of the size of the circuits they are part of and of their relative positions in these circuits. The problems designers are faced with in the area of clock signal distribution, for example, seem to indicate that we have reached the limits in synchronous circuit design. We seek a method that relies neither upon the speed with which a component or its environment responds nor upon the propagation delay of a signal along a connecting wire. Another advantage of this approach is that we have a greater layout freedom, since the lengths of connecting wires are no longer relevant to correctness of operation (although placement definitely does affect speed!).

The other incentive for an asynchronous design approach, reliability, is caused by metastable behavior of many circuits. An arbitration device, i.e. a device that basically grants one out of several requests, is an example of a circuit exhibiting metastable behavior. Under simple continuity assumptions, as described by Hurtado and Elliott (1975) and Marino (1981), it has a metastable state. The closer its initial state is to the metastable state the longer it takes before it settles down in one of its stable states. In clocked systems, where all computational units are assumed to complete each of their computations within a fixed and bounded amount of time, this glitch phenomenon may lead to malfunctioning. This problem was first noticed by

Chaney and Molnar (1973) in the late sixties. The only way to guarantee fully correct communications with an arbitration device seems to be to make the communicating parts asynchronous.

In this paper the foundation of a theory on delay-insensitive systems is laid. The notion of delay-insensitivity is formally defined and a classification is given in an axiomatic way. We briefly touch upon composition of delay-insensitive systems, which has the nice property that delays in interconnecting wires play no role whatsoever in the correctness of the composition. Correctly synthesizing delay-insensitive circuits in the first place is not a topic addressed here. One can follow the method proposed by Seitz (1980) and divide a chip into isochronic regions. These regions are so small that, within a region, the wire delays are negligibly small. Components in such regions are then interconnected by wires with delays about which no assumptions are made. Another method is the one proposed by Molnar et al. (1985). They model a circuit as a Huffman asynchronous sequential circuit with certain of its inputs consisting of the fedback values of some of its outputs. Then it can be shown that the circuit thus obtained is delay-insensitive in its communications with the environment, provided that both the combinational circuit and the internal delays meet certain conditions, e.g. the ones derived by Fang and Molnar (1981), Fang (1983, 1985).

We begin this paper with an introduction to trace theory, as far as we need it for the scope of this paper, and discuss a composition operator, blending, in particular. A more comprehensive discussion can be found in van de Snepscheut (1985). Subsequently, we define and classify delay-insensitive components and illustrate the definitions with a number of examples. It turns out that, with this definition of delay-insensitivity, the specification of the composite of a set of properly composed delay-insensitive components can easily be expressed in terms of the specifications of the composing parts and the blend operator.

Trace theory

In order to define and classify delay-insensitive circuits we need a formalism for their specification. For that purpose we use trace theory, which is a discrete and metric-free formalism. A thorough discussion of trace theory can be found in van de Snepscheut (1985). It should be noted that we present trace theory here in the context of a specific interpretation as a model for circuits.

We consider a mechanism and its environment that communicate with one another by sending and receiving signals. There are two types of signals: from the environment to the mechanism, the *inputs*, and from the mechanism to the environment, which we call *outputs*. We assume that signals are conveyed via (finitely many) wires. With each wire we associate a symbol. A signal via a wire is denoted by the symbol associated with that wire.

A *trace structure* T is a pair $\langle \mathbf{a}T, \mathbf{t}T \rangle$ in which $\mathbf{a}T$ is a finite set of symbols, the *alphabet* of T, and $\mathbf{t}T$ is a set of finite-length sequences of elements of $\mathbf{a}T$, which are called *traces*. The alphabet is the set of symbols that the wires are associated with. The trace set specifies all sequences of communication actions that can take place between the mechanism and its environment. Together with a partitioning of $\mathbf{a}T$ into an input alphabet, $\mathbf{i}T$, and an output alphabet, $\mathbf{o}T$, trace structure T is viewed as the specification of such a mechanism-environment pair.

With a mechanism-environment pair in operation, specified by such a trace structure, we associate a *trace thus far generated*. This is to be a trace of the specifying trace structure. Initially, this trace thus far generated is the empty trace, denoted by ε. In order to satisfy the condition that the trace thus far generated be an element of the specifying trace structure we require a trace set to contain ε. Each act of communication between mechanism and environment corresponds to extending the trace thus far generated with the symbol associated with that act of communication. Apparently, a trace that has a prefix that does not belong to the trace set can never be trace thus far generated. (Trace s is called a *prefix* of trace t if there exists a trace u such that $su=t$.) Therefore, we require such a trace set to be *prefix-closed*, which means that all prefixes of any trace in the trace set belong to that trace set as well.

Example 0. A Muller-C element, or C-element for short (Miller 1965), is an element with two inputs and one output. It is supposed to synchronize the inputs, i.e. after having received an input change on both input wires, it produces a change on the output wire. Its specification is a trace structure with input alphabet $\{a,b\}$, output alphabet $\{c\}$, and trace set $\{\varepsilon, a, b, ab, ba, abc, bac, \ldots\}$.
(End of Example)

The *projection* of trace t on a set of symbols A, denoted by $t\lceil A$, is defined as follows

if $t=\varepsilon$ then $t\lceil A=\varepsilon$
if $t=ua \wedge a\in A$ then $t\lceil A=(u\lceil A)\,a$
if $t=ua \wedge a\notin A$ then $t\lceil A=(u\lceil A)$

Two trace structures can be composed. Composition can, for the time being, be appreciated as the cooperation of two mechanisms where each communication (symbol) in the intersection of the two alphabets represents a joint action of both mechanisms. This leads to the following definition. The *weave* of two trace structures S and T, denoted by $S \mathbf{w} T$, is the trace structure

$$\langle \mathbf{a}S \cup \mathbf{a}T, \{x \in (\mathbf{a}S \cup \mathbf{a}T)^* \mid x \lceil \mathbf{a}S \in \mathbf{t}S \wedge x \lceil \mathbf{a}T \in \mathbf{t}T \} \rangle$$

Notice that weaving amounts to the shuffle operator in Ginsburg (1966) if $\mathbf{a}S \cap \mathbf{a}T = \emptyset$. Notice moreover that delays are not yet taken into account. A communication between mechanism and environment or between two mechanisms is the same for both parties. No distinction is made between sending and receiving.

The *blend* is the second composition operator that we need for trace structures S and T. It is denoted by $S \mathbf{b} T$. It is like the weave but hides the internal communications in the result, i.e. the weave of S and T is projected onto the set of non-common symbols in $\mathbf{a}S$ and $\mathbf{a}T$.

Example 1.

$$\langle \{a, b, d, e\}, \{ab, abe, de\} \rangle \mathbf{w} \langle \{b, c, e, f\}, \{bc, bec, fe\} \rangle = \langle \{a, b, c, d, e, f\}, \{abc, abec, dfe, fde\} \rangle$$

The blend of the two is $\langle \{a, c, d, f\}, \{ac, df, fd\} \rangle$
(End of Example)

From Example 0 one might already have concluded that the representation of a trace set by enumeration of its elements becomes rather cumbersome. Trace structures can be represented in various ways. In Molnar et al. (1985) an *I-net*, a special class of Petri-nets, is used to represent a trace set. In some of the examples in this paper we use (Interface) State Graphs. In most examples we use a representation, called a *command*, that is close to a regular expression, Hopcroft and Ullman (1969). It allows, however, for concurrency to be explicitly expressed. With command S trace structure $\mathbf{TR}(S)$ is associated in the following way.

- A symbol is a command.
  For symbol a $\mathbf{TR}(a) = \langle \{a\}, \{a\} \rangle$.
- If S and T are commands then $(S \mid T)$ is a command.
  $\mathbf{TR}(S \mid T) = \langle \mathbf{a}(\mathbf{TR}(S)) \cup \mathbf{a}(\mathbf{TR}\,T)), \mathbf{t}(\mathbf{TR}(S)) \cup \mathbf{t}(\mathbf{TR}(T)) \rangle$.
- If S and T are commands then $(S; T)$ is a command.
  $\mathbf{TR}(S; T) = \langle \mathbf{a}(\mathbf{TR}(S)) \cup \mathbf{a}(\mathbf{TR}(T)), \{xy \mid x \in \mathbf{t}(\mathbf{TR}(S)) \wedge y \in \mathbf{t}(\mathbf{TR}(T))\} \rangle$.
- If S and T are commands then (S, T) is a command.
  $\mathbf{TR}(S, T) = (\mathbf{TR}(S))\ \mathbf{w}(\mathbf{TR}(T))$.
- If S is a command then S^* is a command.
  $\mathbf{TR}(S^*) = \langle \mathbf{a}(\mathbf{TR}(S)), (\mathbf{t}(\mathbf{TR}(S)))^* \rangle$.

The star has the highest priority, followed by the comma, the semicolon, and the bar. The trace sets thus obtained need not be prefix-closed. Since we are interested in prefix-closed trace sets only, when the command is used for the specification of a mechanism-environment pair we associate with command S the trace structure $\langle \mathbf{a}(\mathbf{TR}(S)), \mathbf{pref}(\mathbf{t}(\mathbf{TR}(S))) \rangle$. For trace set T the set $\mathbf{pref}(T)$ is the set of all prefixes of elements of T.

A command for the C-element of Example 0 could be $(a, b; c)^*$.

Delay-insensitivity

In this section we formally define the notion of delay-insensitivity. First we describe the informal idea behind it, which is due to Molnar et al. (1985). Recall that a trace structure, as introduced in the previous section could be appreciated as the specification of a mechanism-environment pair. A symbol in the alphabet of a trace structure was used to indicate a communication action between mechanism and environment. Due to not taking wire delays into account we could associate one trace thus far generated with such a mechanism-environment pair in operation.

When dealing with delays, however, we have to realize that input symbols represent signals that are under control of the environment and that travel from environment to mechanism, and that output symbols represent signals that are under control of the mechanism and that travel from mechanism to environment. Now, during operation, there are in fact two traces thus far generated: one at the mechanism's boundary and one at the environment's boundary. Because of the delays, they need not be the same, although they are certainly related; a signal originating from one boundary will eventually arrive at the other, and a signal arriving at one boundary must previously have been produced at the other.

As before, a trace thus far generated must be element of the specifying trace set. The rules for extending a trace with a symbol that is under control of the side for which that trace is the trace thus far generated is the same as before: the trace extended with that symbol must be element of the

specifying trace set. In this case we say that a signal is produced or sent.

The obvious counterpart for signals arriving at a boundary is that its trace thus far generated is extended with the symbol associated with that signal. But what if the trace extended with that symbol is not an element of the specifying trace set? This is interpreted as specifying that the receiving end is not (yet) ready to receive that signal and, hence, that something might go wrong if it arrives. (Notice that this problem does not occur when delays are zero, since the two traces thus far generated are equal in that case.) This phenomenon, a signal arriving when the receiving end is not ready for it, is called *computation interference.*

Another phenomenon that we want to avoid is *transmission interference.* This occurs when two signals on the same wire interfere. A specification is called *delay-insensitive* if, when both mechanism and environment act according to that specification in the way outlined above, for all possible pairs of traces thus far generated absence of computation and transmission interference is guaranteed. In Molnar et al. (1985) this idea is referred to as the Foam Rubber Wrapper Postulate: the mechanism is thought of as being wrapped in foam rubber of which the boundaries are flexible and possibly changing: it is immaterial where the communications between mechanism and environment are specified.

It should be noted that this is not the only way to formalize the notion of delay-insensitivity. A slightly different approach has been taken by Seitz (1980) and has been further elaborated by Martin (1985). In their model the specification applies to the module boundary and it is called delay-insensitive if that module can, in principle, communicate in a correct way with its environment irrespective of delays in interconnecting wires. The trace thus far generated at the environment boundary is not necessarily element of the trace structure specifying the module. On the other hand, any trace at the module boundary implied by a trace thus far generated at the environment boundary is element of the specifying trace structure. Hence, in their model absence of computation interference is guaranteed at the module boundary but not at the environment boundary. Therefore, when composing modules with one another, i.e. replacing part of their environments with the other modules, we still have to take into account delays in interconnecting wires, which complicates the definition of an algebraic composition operator expressing this composition. Taking the Foam Rubber Wrapper approach, however, it can be shown that wire delays can be neglected and that communications may be perceived as instantaneous: the blend of the specifying trace structures yields the specification of the resulting composite.

Restrictions that guarantee a trace structure to be delay-insensitive can be captured in four rules. These requirements basically amount to the absence of ordering between certain symbols: the presence of certain traces in a trace structure's trace set implies the presence of other traces in that set. The following situations can be distinguished.

As mentioned before, we want to guarantee absence of transmission interference. Since we do not want to make any assumptions about absolute or relative wire delays we have to limit the number of consecutive transitions on a wire to at most one. In terms of trace structures, where signals via the same wire are represented by the same symbol, this gives the following necessary condition for trace structure T.

$\mathbf{R}_0$ for trace s and symbol $a \in \mathbf{a}T$ $\quad saa \notin \mathbf{t}T$

Two signals being sent one after the other in the same direction via different wires need not be received in the order in which they are sent. In other words, we cannot assume that the order of our communications is preserved. Consequently, a specification of a delay-insensitive component must not express a dependence upon the order in which signals traveling in the same direction are sent or received. Therefore, a trace structure containing a trace with two adjacent symbols of the same type (input or output) also contains the trace with these two symbols swapped. In fact, we conceive adjacent symbols of the same type as not being ordered at all. (Their occurrence as adjacent symbols in a trace is just a shortcoming of our writing in a linear way.) For trace structure T this is expressed by the following restriction.

$\mathbf{R}_1$ for traces s and t and for symbols $a \in \mathbf{a}T$ and $b \in \mathbf{a}T$ of the same type $\quad sabt \in \mathbf{t}T = sbat \in \mathbf{t}T$

Signals in opposite directions are subject to restrictions as well. As opposed to signals of the same type they may have a causal relationship and, hence, have an order. If, however, in some phase of the computation they are not ordered, meaning that for some trace s and symbols a and b both $sab \in \mathbf{t}T$ and $sba \in \mathbf{t}T$, then the order should, intuitively at least, be immaterial. This results for trace structure T in the rule

$\mathbf{R}_2$ for traces s and t, and for symbols $a \in \mathbf{a}T$ and $b \in \mathbf{a}T$ of different types
$(sab \in \mathbf{t}T \wedge sba \in \mathbf{t}T) \Rightarrow (sabt \in \mathbf{t}T = sbat \in \mathbf{t}T)$

Finally, we observe the following. A mechanism ready to receive a certain signal from its environment, which means that its trace thus far generated extended with that symbol belongs to the trace set, must not change its readiness by sending a signal to its environment. The signal that the mechanism seemed to be willing to receive may be on its way already and cannot be revoked. In other words, we cannot allow a symbol to disable a symbol of another type. Symbol *a disables* symbol *b* in trace structure T if there is a trace s with $sa \in \mathbf{t}T \wedge sb \in \mathbf{t}T \wedge sab \notin \mathbf{t}T$. Symbols of the same type may disable one another, however. If these symbols are input symbols then the environment has to make a decision as to which symbol(s) to send. If, on the other hand, the symbols are output symbols then the mechanism has to make that decision. Since a correct use of arbitration devices is one of the important incentives to the study of delay-insensitive circuits, the various types of decisions are a key to the classification. Three classes, each of them described by one of the following non-disabling rules, can be distinguished. For trace structure T we have

$\mathbf{R}'_3$ for trace s and distinct symbols $a \in \mathbf{a}T$ and $b \in \mathbf{a}T$
$sa \in \mathbf{t}T \wedge sb \in \mathbf{t}T \Rightarrow sab \in \mathbf{t}T$

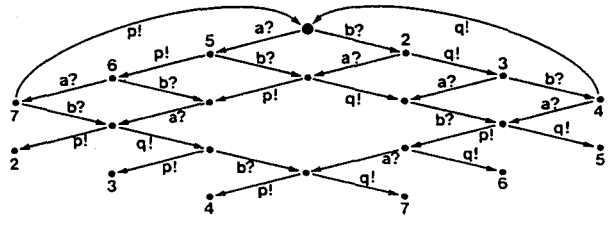
$\mathbf{R}''_3$ for trace s and distinct symbols $a \in \mathbf{a}T$ and $b \in \mathbf{a}T$, not both input symbols, $sa \in \mathbf{a}T \wedge sb \in \mathbf{t}T \Rightarrow sab \in \mathbf{t}T$

$\mathbf{R}'''_3$ for trace s and symbols $a \in \mathbf{a}T$ and $b \in \mathbf{a}T$ of different types
$sa \in \mathbf{t}T \wedge sb \in \mathbf{t}T \Rightarrow sab \in \mathbf{t}T$

All delay-insensitive trace structures satisfy rules 0 and 1. The class satisfying rules 2 and 3′ as well is called the *synchronization class*. It is also denoted by $\mathbf{C}_1$. A specification in this class allows for synchronization only. Due to the absence of decisions, no data transmission is possible. An example of such a specification is the C-element of Example 0. The class allowing for input symbols to be disabled, satisfying therefore rules 2 and 3″, is called the *data communication class*. It is also denoted by $\mathbf{C}_2$. Here the data is encoded by means of the possible decisions. Finally, we have $\mathbf{C}_3$, or the *arbitration class*, which allows a mechanism to choose between two output symbols. Specifications in this class satisfy, in addition to rules 0 and 1, rules 2 and 3‴. Obviously, $\mathbf{C}_1 \subseteq \mathbf{C}_2 \subseteq \mathbf{C}_3$.

Example 2. A binary variable is a mechanism that can store one bit of information, which may be retrieved afterwards on request an unbounded number of times. The mechanism has input alphabet $\{x_0, x_1, b\}$, output alphabet $\{y_0, y_1, a\}$, and command $(x_0; a; (b; y_0)^* | x_1; a; (b; y_1)^*)^*$. Symbol a acknowledges the reception of a bit (either x_0 or x_1), and b is the request for the currently stored value. Initially, a choice has to be made between x_0 and x_1. Afterwards there is a choice to be made between b, x_0, and x_1. Since these symbols are input symbols, the specification is a $\mathbf{C}_2$.
(End of Example)

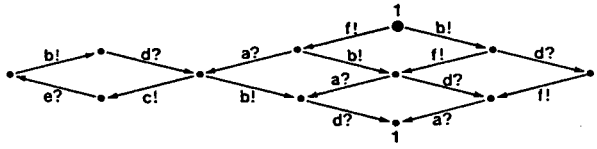
Example 3. An arbiter is a device that grants one out of two requests. The arbiter described in this example can serve two processes that share a resource. Typically, a process issues a request for access to the resource towards the arbiter and waits until that request has been granted by the arbiter. Then, after having accessed the resource, a process acknowledges the release of the resource, upon which the arbiter may signal to that process that a next request may be issued. For issuing a request and for signaling the release of the resource we use input symbols a and b. Symbols p and q are used for granting access to the resource, p upon request a, and q upon request b, and for signaling a process that a next request may be issued. Rather than by a command we represent this trace structure by a state graph. A state graph is a directed graph with one special node, the start node, and arcs labeled with the symbols of the trace structure's alphabet. Each path from the start node corresponds to a trace, viz. the one that is brought about by the labels of the consecutive arcs in that path. A state graph is said to represent a trace structure if it has the same trace set as that trace structure. Rules 0 through 3 are usually more easily checked in a state graph than in a command. The start node is drawn fat. For clarity we attach question marks to arcs labeled with an input symbol and exclamation marks to arcs labeled with an output symbol. Nodes that have been attached the same number are identical. The state graph for the arbiter described above, from which it can easily be seen that it is a $\mathbf{C}_3$, is



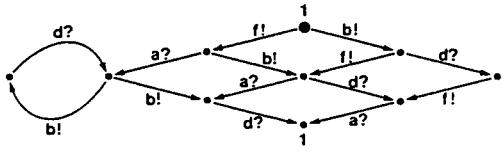
(End of Example)

Example 4. The mechanism in this example is used to demonstrate that the intuition used to derive

rule 2, viz. that the order of concurrent signals of different types is immaterial, is wrong. The trace structure defined here has input alphabet $\{a, d, e\}$, output alphabet $\{b, c, f\}$, and state graph



Apparently, if the mechanism sends signal c it is able and will remain able to receive signal e. Although we refrain from composition issues here, it should be clear that this mechanism may be composed with a mechanism that sends signal e upon every reception of signal c. The resulting specification has state graph



which does not belong to any of the classes defined, although the original trace structure was a $\mathbf{C}_3$. It violates rule 2: traces fab and fba belong to the trace set and so does $fabdbdb$. Trace $fbadbdb$ does not belong to the trace set, however.
(End of Example)

To remedy the anomaly described in Example 4 we have to analyze the situation of two concurrent symbols of different types more carefully.

If at the environment's boundary the order between two input and output signals that can occur in either order according to the specification is input-before-output (input and output are with respect to the mechanism), nothing can be said about their order at the mechanism's boundary. If, on the other hand, the order between such signals is output-before-input at the environment's boundary, the same order between these symbols is implied at the other boundary. The first situation, i.e. input-before-output at the environment's boundary is the only one that forces other traces to be in the trace set as well.

Assume that we have, for traces s and t, and for input symbol a and output symbol b, $sabt \in \mathbf{t}T$ and $sbat \in \mathbf{t}T$. Trace $sabt$ can be the trace thus far generated at the environment's boundary when trace $sbat$ is the one at the mechanism's boundary. Now, if $sabt$ can be extended, according to the trace set, with input symbol c, which means that signal c may be sent from the environment to the mechanism, then a necessary condition for absence of computation interference at the mechanism's boundary is the presence of trace $sbatc$ in the trace set. Due to symmetry between mechanism and environment, a similar observation applies to an output-before-input order of concurrent signals at the mechanism's boundary and an input-before-output order at the other one. In this case an output signal possible at the mechanism's boundary should be possible at the environment's boundary as well. This results in the following rule

$\mathbf{R}'_2$ for traces s and t, and for symbol $b \in \mathbf{a}T$ of another type than symbols $a \in \mathbf{a}T$ and $c \in \mathbf{a}T$
$sabtc \in \mathbf{t}T \wedge sbat \in \mathbf{t}T \Rightarrow sbatc \in \mathbf{t}T$

The class of trace structures that satisfy rules 0, 1, 2′, and 3‴ is called $\mathbf{C}_4$ or the *class of delay-insensitive trace structures*. Notice that $\mathbf{R}_2 \Rightarrow \mathbf{R}'_2$ and that therefore $\mathbf{C}_3 \subseteq \mathbf{C}_4$. It can be proved that the rules derived here for a $\mathbf{C}_4$ are not only necessary but also sufficient, i.e. if an environment and a mechanism communicate with one another according to a delay-insensitive trace-structure then absence of computation and transmission interference is guaranteed. The proof is very long and, hence, omitted here. It can be found in Udding (1984).

Fairness and delay-insensitive arbiters

To illustrate the theory developed in the previous section and to show its use, we prove that, with delay-insensitivity as defined above and with fairness as defined below, no specification for a fair delay-insensitive arbiter exists. Fair delay-insensitive arbiters do exist when the definition of delay-insensitivity is chosen to be more liberal, as Martin (1985) does, or if the definition of fairness is chosen to be more liberal, as Black (1985) does. In the latter paper an overview is given of the possible notions of fairness. In this paper an arbiter is called unfair if it always can grant access to the resource to one process while denying the other process access to the resource indefinitely. Notice that the arbiter of Example 3 is unfair for that reason: all traces in $b(ap)^*$ are elements of that trace set.

Let us first take a look at the basic functions that an arbiter has to satisfy. Like in Example 3 we consider an arbiter that can serve two processes. Requests are issued via a and b, and the grants are represented by p and q respectively. Without loss of generality these symbols are used for signaling the release of the resource and the subsequent acknowledgement thereof from the arbiter as well.

The first requirement to be imposed upon an arbiter is that the two processes contending for

the resource are allowed to do so independently. That is, a grant for access may be followed by a release signal at any time and, likewise, if the arbiter signals its readiness for the next request, that request may be issued at any time after that ready signal. A more general and formal discussion of independence may be found in Udding (1984).

The second requirement, which has to do with fairness, is that a process is not unnecessarily deprived of a grant. One can argue about the formalization of this statement, but part of it is certainly that a process will be granted access to the resource upon every request and will be allowed to issue a next request, as long as the other process has not issued any request for access to the resource. Together with the independence requirement mentioned above, any alternation of a's and p's, not starting with p, should be a trace belonging to the trace set of an arbiter. Using regular expressions, this implies that for any integer n, $n \geq 0$, $(ap)^n$ belongs to the trace set of an arbiter.

We show that any delay-insensitive trace structure that satisfies the two conditions mentioned above is unfair, meaning that all traces of the form $b(ap)^k$ for $k \geq 1$ are forced to be in the specifying trace set on account of the rules for delay-insensitivity. By mathematical induction on k we prove that for all integers k, $k \geq 1$, and n, $n \geq 0$, $(ap)^n b(ap)^k \in \mathbf{t}T$ for any delay-insensitive trace structure T that satisfies the two conditions.

Base. $k = 1$. *Let* $n \geq 0$.

$$
\begin{aligned}
& \textit{true} \\
= \; & \{\text{fairness and independence assumption}\} \\
& (ap)^{n+1} \in \mathbf{t}T \\
\Rightarrow \; & \{\mathbf{t}T \text{ is prefix-closed}\} \\
& (ap)^n ap \in \mathbf{t}T \wedge (ap)^n a \in \mathbf{t}T \\
\Rightarrow \; & \{\text{independence assumption}\} \\
& (ap)^n ap \in \mathbf{t}T \wedge (ap)^n ab \in \mathbf{t}T \\
\Rightarrow \; & \{\text{rule 3}''' \text{ of delay-insensitivity}\} \\
& (ap)^n abp \in \mathbf{t}T \\
\Rightarrow \; & \{\text{rule 1 of delay-insensitivity}\} \\
& (ap)^n bap \in \mathbf{t}T
\end{aligned}
$$

Step. Assume for all n, $n \geq 0$, and for some k, $k \geq 1$, that we have $(ap)^n b(ap)^k \in \mathbf{t}T$. Then we derive for all n, $n \geq 0$,

$$
\begin{aligned}
& \textit{true} \\
= \; & \{\text{induction hypothesis}\} \\
& (ap)^{n+1} b(ap)^k \in \mathbf{t}T \wedge (ap)^n b(ap)^k \in \mathbf{t}T \\
\Rightarrow \; & \{\text{independence assumption, using } k \geq 1\} \\
& (ap)^{n+1} b(ap)^k \in \mathbf{t}T \wedge (ap)^n bap(ap)^{k-1} a \in \mathbf{t}T \\
\Rightarrow \; & \{\text{rule 1 of delay-insensitivity, using } n \geq 0 \text{ and } k \geq 1\} \\
& (ap)^n apb(ap)^{k-1} ap \in \mathbf{t}T \\
& \wedge (ap)^n abp(ap)^{k-1} a \in \mathbf{t}T \\
\Rightarrow \; & \{\text{rule 2}' \text{ of delay-insensitivity}\} \\
& (ap)^n abp(ap)^{k-1} ap \in \mathbf{t}T \\
\Rightarrow \; & \{\text{rule 1 of delay-insensitivity}\} \\
& (ap)^n bap(ap)^{k-1} ap \in \mathbf{t}T \\
= \; & \{\text{calculus}\} \\
& (ap)^n b(ap)^{k+1} \in \mathbf{t}T
\end{aligned}
$$

Concluding remarks

In this paper we have introduced the notions of transmission and computation interference. Moreover, we have stated necessary and sufficient conditions for a circuit specification to be called delay-insensitive. Absence of computation and transmission interference is guaranteed and a circuit will behave according to its specification whatever the delays in the wires connecting the circuit to its environment are. Although composition of circuits is not a topic discussed in this paper, we mention that this formal approach can be used for composition of circuits as well and that the specification of the composite can be expressed as the blend of the specifications of the composing parts. Hence, starting with the proper (i.e. delay-insensitive) components, we can treat communication actions as instantaneous.

Acknowledgements. The work presented here would have been impossible without the help and cooperation of numerous people. First of all I am very grateful to Charles E. Molnar, who made my visit to the Computer Systems Laboratory at Washington University, St. Louis, possible, where I became familiar with delay-insensitive systems and benefited a lot from the approach towards delay-insensitivity that had been established already in his group. Especially the insight that a specification pertains symmetrically to mechanism and environment was very helpful. Moreover, I am greatly indebted to Ting-Pien Fang, with whom I had very many discussions that tremendously improved my understanding of the subject. Finally, I want to thank both the Eindhoven Tuesday Afternoon Club and the Eindhoven VLSI Club, especially Jan L.A. van de Snepscheut and Martin Rem who introduced me to trace theory and made me aware of its possible applications to delay-insensitive systems. Financial support for portions of this work was provided by Grant 1-R24-RR01379 of the National Institutes of Health.

Charles E. Molnar, David L. Black, and the Eindhoven VLSI Club are gratefully acknowledged for their comments on an earlier version of this manuscript.

References

Black DL (1985) On the existence of delay-insensitive fair arbiters: trace theory and its limitations. Technical Memorandum CMU-CS-85-173, Carnegie-Mellon University, Pittsburgh, PA (October)

Chaney TJ, Molnar CE (1973) Anomalous behaviour of synchronizer and arbiter circuits. IEEE Trans Comput, C-22:421–422

Fang, TP, Molnar CE (1983) Synthesis of reliable speed-independent circuit modules: II. Circuit and delay conditions to ensure operation free of problems from races and hazards. Technical Memorandum 298, Computer Systems Laboratory, Washington University, St. Louis, MO (December)

Fang TP (1985) Prevention of problems caused by distribution of delays in clock-free realizations of modules of delay-insensitive systems. Technical Memorandum in Preparation, Computer Systems Laboratory, Washington University, St. Louis, MO (December)

Ginsburg S (1966) The mathematical theory of context-free languages. McGraw-Hill

Hopcroft JE, Ullman JD (1969) Formal languages and their relation to automata. Addison-Wesley, London Amsterdam Paris

Hurtado ME, Elliott DL (1975) Ambiguous behavior of logic bistable systems. Proceedings of the 13th Annual Allerton Conference on Circuit and System Theory, Champaign-Urbana, Ill., pp 605–611

Marino LR (1981) General theory of metastable operation. IEEE Transactions on Computers, C-30, 2:107–115

Martin AJ (1985) A delay-insensitive fair arbiter. Technical Memorandum 5193:TR:85, Comput Sci Dep, California Institute of Technology, Pasadena, CA

Miller RE (1965) Speed independent switching circuit theory. In: Switching theory. Sequential circuits and machines, vol. II, chap 10. Wiley, New York

Molnar CE, Fang T (1981) An asynchronous system design methodology. Technical Memorandum 287, Computer Systems Laboratory, Washington University, St. Louis, MO

Molnar CE, Fang T, Rosenberger FU (1985) Synthesis of delay-insensitive modules. 1985 Chapel Hill Conference on VLSI, Chapel Hill, NC, pp 67–86 (May)

Seitz CL (1971) Self-timed VLSI systems. Proceedings of the Caltech Conference on VLSI, Pasadena, CA, pp 345–355

Seitz CL (1980) System timing. In: Mead CA, Conway LA Introduction to VLSI Systems, chap 7. Addison Wesley, London Amsterdam Paris

van de Snepscheut JLA (1985) Trace theory and VLSI design. Lecture Notes Comput Sci 200, Springer, Berlin Heidelberg New York Tokyo

Udding JT (1984) Classification and Composition of Delay-Insensitive Circuits. Doctoral Dissertation, Eindhoven University of Technology, Eindhoven, Netherlands (September)