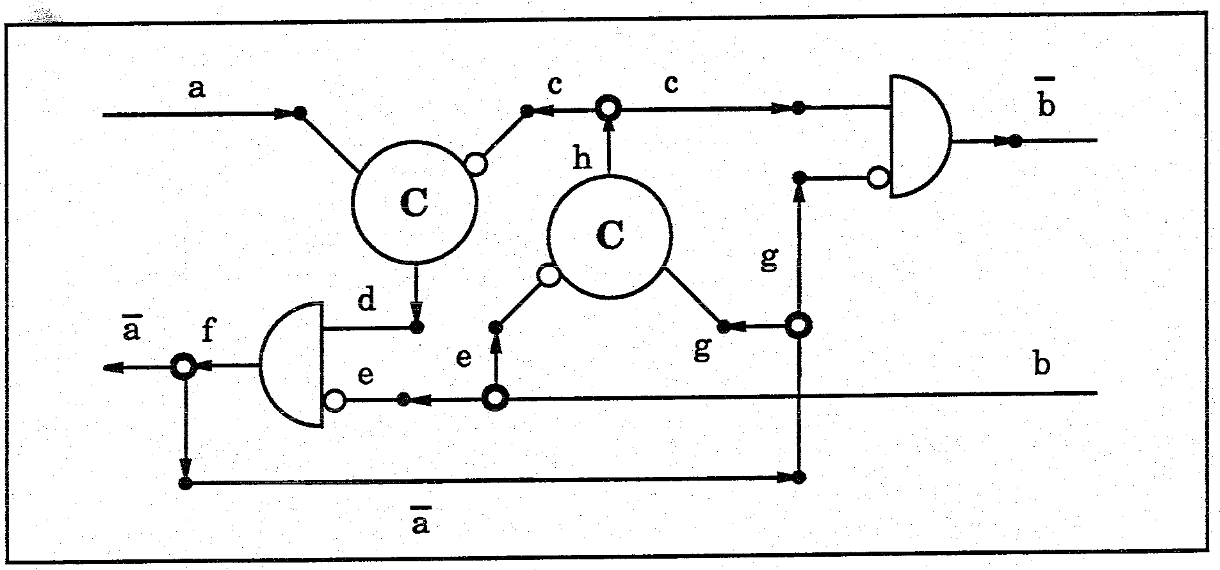
Specification and verification of a circuit in ACP (revised version)



Specification and Verification of a circuit in ACP (revised version)

J.C.M. Baeten
F.W. Vaandrager

Report P8821

October 1988

University of Amsterdam
Department of Mathematics and Computer Science
Programming Research Group

Specification and verification of a circuit in ACP (revised version)

J.C.M. Baeten
F.W. Vaandrager

Report P8821

October 1988

J.C.M. Baeten

Programming Research Group
Department of Mathematics and Computer Science
University of Amsterdam

NIKHEF-K A108
Kruislaan 409
1098 SJ Amsterdam

P.O. Box 41882
1009 DB Amsterdam
The Netherlands

tel. +31 20 5922012

F.W. Vaandrager

Department of Software Technology
Centre for Mathematics and Computer Science

CWI M337
Kruislaan 413
1098 SJ Amsterdam

P.O. Box 4079
1009 AB Amsterdam
The Netherlands

tel. +31 20 5924125

Universiteit van Amsterdam, 1988

Specification and verification of a circuit in ACP
(revised version)

Jos C.M. Baeten
*Programming Research Group, University of Amsterdam,
P.O.Box 41882, 1009 DB Amsterdam, The Netherlands.*

Frits W. Vaandrager
*Department of Software Technology, Centre for Mathematics and Computer Science,
P.O.Box 4079, 1009 AB Amsterdam, The Netherlands.*

A simple circuit is specified and verified in the framework of the Algebra of Communicating Processes (ACP). The usefulness of the priority operator for this type of applications is demonstrated.

1980 Mathematical Subject Classification (1985 revision): 68M10, 68Q35, 68Q60, 94C99.
1987 CR Categories: B.6.1, B.7.1, F.1.1, F.3.2.
Key words & Phrases: process algebra, concurrency, circuit, interference, priority operator, put mechanism.
Note: Partial support received from the European Communities under ESPRIT contract 432, An Integrated Formal Approach to Industrial Software Development (Meteor).

1. INTRODUCTION.

ACP, the Algebra of Communicating Processes of BERGSTRA & KLOP [BK1, BK2, BK3], is an algebraic framework designed both for the specification and for the verification of concurrent systems. The ACP framework is closely related to MILNER's CCS [MI] and HOARE's CSP [H]. In his seminal work [MI], Milner already mentioned hardware description as a possible application area of his calculus. Yet most applications of theories like CCS, CSP and ACP are in fields as protocol verification, semantics of programming languages and distributed algorithms. Only a group around REM [R] has been working actively on VLSI circuits in the setting of *trace theory*, which is inspired by an early variant of CSP.

The subject of this paper is the specification and verification of a simple circuit in the ACP framework. The description of the circuit we consider here is derived from KALDEWAIJ [K].

In the example we will encounter an interesting application of the priority operator θ, introduced in BAETEN, BERGSTRA & KLOP [BBK1]. The application arises because wires in a circuit have a *direction*. If, at one side of the wire, a component takes the initiative to change the value on the wire, this just happens. This means that if we model a circuit in a formalism with synchronous communication, we have to guarantee that whenever one process wants to perform a send-action, corresponding to a change in the value of a wire, the communication partner is always willing to perform the associated read-action. In this report we will show how this fundamental correctness criterion (called *absence of interference*) can be formulated by means of the put mechanism of BERGSTRA [B], using the priority operator.

This paper can be viewed as the result of an integration exercise: we tried to incorporate notions from trace theory into the ACP formalism, and moreover investigated whether notions from ACP can be used to enhance the trace theoretic modelling of circuits.

Although we tried to keep this paper as much self-contained as possible, this is not an introductory paper on process algebra. For a survey of the ACP formalism we refer the reader to BERGSTRA & KLOP [BK3], or to [BK1], where also a comparison with related approaches can be found.

THE ALGEBRA OF COMMUNICATING PROCESSES

The axiomatic framework in which we present this paper is ACP, the Algebra of Communicating Processes, as described in [BK1, BK3]. Here, we give a brief review of ACP.

Process algebra starts from a finite collection A of given objects, called atomic actions, atoms or steps. These actions are taken to be indivisible, usually have no duration and form the basic building blocks of our systems. The first two compositional operators we consider are $\cdot$, denoting sequential composition, and + for alternative composition. If x and y are two processes, then $x \cdot y$ is the process that starts the execution of y after the completion of x, and $x+y$ is the process that chooses either x or y and executes the chosen process. Each time a choice is made, we choose from a set of alternatives. We do not specify whether the choice is made by the process itself, or by the environment. Axioms A1-5 in table 1 below give the laws that + and $\cdot$ obey. We leave out $\cdot$ and brackets as in regular algebra, so $xy + z$ means $(x \cdot y) + z$.

On intuitive grounds $x(y + z)$ and $xy + xz$ present different mechanisms (because the moment of choice is different), and therefore an axiom $x(y + z) = xy + xz$ is not included.

We have a special constant δ denoting deadlock, the acknowledgement of a process that it cannot do anything anymore, the absence of an alternative. Axioms A6,7 give the laws for δ. Together, the axioms A1-A7 are referred to as BPA, which stands for Basic Process Algebra.

Next, we have the parallel composition operator $\|$, called merge. The merge of processes x and y will interleave the actions of x and y, except for the communication actions. In $x \| y$, we can either do a step from x, or a step from y, or x and y both synchronously perform an action, which together make up a new action, the communication action. This trichotomy is expressed in axiom CM1. Here, we use two auxiliary operators $\lfloor\!\lfloor$ (left-merge) and $|$ (communication merge). Thus, $x \lfloor\!\lfloor y$ is $x \| y$, but with the restriction that the first step comes from x, and $x | y$ is $x \| y$ with a communication step as the first step. Axioms CM2-9 give the laws for $\lfloor\!\lfloor$ and $|$. We assume the communication function is given on atomic actions and obeys laws C1-3.

EXAMPLES:

$a \| b = a \lfloor\!\lfloor b + b \lfloor\!\lfloor a + a | b = ab + ba + a | b;$

$(ab) \lfloor\!\lfloor c = a(b \| c) = a(bc + cb + b | c);$

$(ab) | (cd) = (a | c)(b \| d) = (a | c)(bd + db + b | d).$

Finally, on the left-hand side of table 1 we have the laws for the encapsulation operator ∂_H. Here H is a set of atoms, and ∂_H blocks actions from H by renaming them into δ. The operator ∂_H can be used to encapsulate a process, i.e. to block communications with the environment.

EXAMPLE:
Suppose $a \mid b = c$, $c \neq a$, $c \neq b$, $H = \{a, b\}$, then $\partial_H(a \| b) = \partial_H(ab + ba + a \mid b) = \delta\delta + \delta\delta + c =$
$= \delta + \delta + c = c$.

In the following table we have $a,b,c \in A \cup \{\delta\}$, x,y,z are arbitrary processes, and $H \subseteq A$.

$x + y = y + x$	A1	$x \| y = x \lfloor\!\lfloor y + y \lfloor\!\lfloor x + x \mid y$	CM1
$x + (y + z) = (x + y) + z$	A2	$a \lfloor\!\lfloor x = ax$	CM2
$x + x = x$	A3	$ax \lfloor\!\lfloor y = a(x \| y)$	CM3
$(x + y)z = xz + yz$	A4	$(x + y) \lfloor\!\lfloor z = x \lfloor\!\lfloor z + y \lfloor\!\lfloor z$	CM4
$(xy)z = x(yz)$	A5	$ax \mid b = (a \mid b)x$	CM5
$x + \delta = x$	A6	$a \mid bx = (a \mid b)x$	CM6
$\delta x = \delta$	A7	$ax \mid by = (a \mid b)(x \| y)$	CM7
		$(x+y) \mid z = x \mid z + y \mid z$	CM8
		$x \mid (y+z) = x \mid y + x \mid z$	CM9
$\partial_H(a) = a$ if $a \notin H$	D1		
$\partial_H(a) = \delta$ if $a \in H$	D2	$a \mid b = b \mid a$	C1
$\partial_H(x + y) = \partial_H(x) + \partial_H(y)$	D3	$(a \mid b) \mid c = a \mid (b \mid c)$	C2
$\partial_H(xy) = \partial_H(x) \cdot \partial_H(y)$	D4	$\delta \mid a = \delta$	C3

Table 1. ACP $(a,b,c \in A \cup \{\delta\}, H \subseteq A)$.

The language ACP can serve to give specifications of systems. However, if we want to do *verifications*, we have need for an *abstraction* mechanism, for, in order to verify that a given implementation satisfies the required external behavior, we need to abstract from all internal actions of the system. Here, we use the contstant τ of MILNER [MI] for this purpose, added to the system ACP as expounded in BERGSTRA & KLOP [BK2]. The axiom system ACP_τ consists of the axioms of ACP together with the axioms in table 2 below. For more information, we refer to [BK2].

$x\tau = x$	T1	$\tau \lfloor\!\lfloor x = \tau x$	TM1
$\tau x + x = \tau x$	T2	$\tau x \lfloor\!\lfloor y = \tau(x \| y)$	TM2
$a(\tau x + y) = a(\tau x + y) + ax$	T3	$\tau \mid x = \delta$	TC1
		$x \mid \tau = \delta$	TC2
$\tau_I(\tau) = \tau$	TI1	$\tau x \mid y = x \mid y$	TC3
$\tau_I(a) = a$ if $a \notin I$	TI2	$x \mid \tau y = x \mid y$	TC4
$\tau_I(a) = \tau$ if $a \in I$	TI3		
$\tau_I(x + y) = \tau_I(x) + \tau_I(y)$	TI4	$\partial_H(\tau) = \tau$	DT
$\tau_I(xy) = \tau_I(x) \cdot \tau_I(y)$	TI5		

Table 2. ACP_τ $(a \in A \cup \{\delta\}, I \subseteq A)$.

2. BASIC ELEMENTS.

A *circuit* consists of a number of *basic elements*, connected by *wires*. On each wire there is either a *high voltage* (1) or a *low voltage* (0). The state of the circuit is characterized by the voltages on its wires. In the process algebra modeling we view the basic elements as processes, and with each wire we associate the name of a communication port. For every x from a given set P of port names, the alphabet A contains atomic actions $sx\uparrow$, $sx\downarrow$, $rx\uparrow$, $rx\downarrow$, $cx\uparrow$ and $cx\downarrow$. Communication is defined by $sx\uparrow \mid rx\uparrow = cx\uparrow$ and $sx\downarrow \mid rx\downarrow = cx\downarrow$. Whenever an action $cx\uparrow$ takes place the interpretation is that the voltage on wire x becomes 1, and whenever an action $cx\downarrow$ takes place, the voltage becomes 0. At a slightly higher level of abstraction, we make no distinction between actions $cx\uparrow$ and $cx\downarrow$ (and also not between $sx\uparrow$ and $sx\downarrow$, or $rx\downarrow$ and $rx\downarrow$), calling both cx. We use symbols x,y,z for the names of ports and symbols p,q,r for voltages.

In the sequel, we describe the basic components *inverter*, *And-element*, *C-element*, and *fork*.

2.1. THE INVERTER.

The *inverter* has one input wire and one output wire. The output wire always has a different voltage than the input wire. In the initial state, the input wire x has voltage 0, the output wire y voltage 1. The graphical representation is shown in fig. 1.

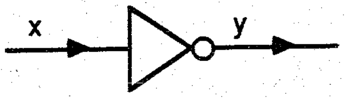


Fig. 1.

The inverter is recursively specified as follows:

$$I^0_{xy} = rx\uparrow \cdot sy\downarrow \cdot I^1_{xy}$$
$$I^1_{xy} = rx\downarrow \cdot sy\uparrow \cdot I^0_{xy}.$$

Like all other processes that will be associated with basic elements, the process I^0_{xy} obeys the following two rules.

1. For a given port, the $\uparrow$ and $\downarrow$ actions alternate.

The reason for this rule becomes obvious when we look at the interpretations of the $\uparrow$ and $\downarrow$ actions. An $\uparrow$-action occurs whenever the voltage on the wire becomes 1, whereas an $\downarrow$-action occurs whenever the voltage becomes 0. Because we assumed that on each wire the voltage is either 1 or 0, there must always be an $\downarrow$-action between two occurrences of an $\uparrow$-action: if the voltage is 1, it must become 0 before it can become 1 again. Analogously, there must always be an $\uparrow$-action between two occurrences of an $\downarrow$-action.

The second rule is less obvious:

2. Actions on input and output ports are never enabled simultaneously.

This means that, for example, we do not have equations for the inverter process of the form

$$I^0_{xy} = rx\uparrow \cdot (sy\downarrow \cdot I^1_{xy} + rx\downarrow \cdot I^0_{xy}).$$

When we look at physical inverters, it is of course possible that, in the starting state, the voltage of the input wire becomes 1, and then, before the voltage on the output wire has been changed to 0, immediately becomes 0 again. The problem is that it is not clear what will happen to the voltage of the output wire in situations like this. It might be the case that the voltage changes on the input wire occur so fast that the changes are not propagated through the inverter so that the output voltage remains unchanged. Another possibility is that the voltage changes on the input are somehow kept in memory, so that, sooner or later, both changes will occur on the output wire.

Clearly, it will be very difficult (if not impossible) to model phenomena like this in process algebra. Therefore, we do not take into account the possibility that voltage changes occur on the input ports, if voltage changes on the output ports are possible. If a basic component is placed in an environment that offers premature voltage changes on the input ports, we speak of *interference*. One may say that such circuits are not well-designed. We will say more about interference in section 3.

2.2. THE AND-ELEMENT.

The *And-element* has two input wires and one output wire. If both inputs have voltage 1 the output will have voltage 1, otherwise the output will have voltage 0. The graphical representation of the And-element is as shown in fig. 2.

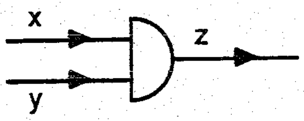


Fig. 2.

The behavior of an And-element with input wires x and y and output wire z will be given by a specification with variables A^{pq}_{xyz}, where p is the voltage of wire x and q the voltage of wire y. In state A^{pq}_{xyz}, the voltage of wire z is p·q.

$$A^{00}_{xyz} = rx\uparrow\cdot A^{10}_{xyz} + ry\uparrow\cdot A^{01}_{xyz}$$
$$A^{10}_{xyz} = rx\downarrow\cdot A^{00}_{xyz} + ry\uparrow\cdot sz\uparrow\cdot A^{11}_{xyz}$$
$$A^{01}_{xyz} = rx\uparrow\cdot sz\uparrow\cdot A^{11}_{xyz} + ry\downarrow\cdot A^{00}_{xyz}$$
$$A^{11}_{xyz} = rx\downarrow\cdot sz\downarrow\cdot A^{01}_{xyz} + ry\downarrow\cdot sz\downarrow\cdot A^{10}_{xyz}.$$

We can incorporate an inverter with an And-element, graphically represented as in fig. 3. We call this element A^{10}_{xyz}.

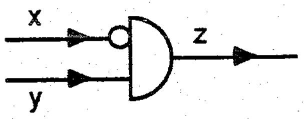


Fig. 3.

Such an And-element has the same specification as a regular And-element, with two differences: first, all arrows of actions on port x are reversed; second, when the voltages on wires x,y,z are all 0, it is in initial state $A^{10}_{\underline{x}yz}$. Note that the process $A^{10}_{\underline{x}yz}$ is not exactly the same as the composition of an inverter and an And-element (connected via a port w, so $\tau_C \circ \partial_H(I^0_{xw} \parallel A^{10}_{\underline{w}yz})$ with $H = \{rw\uparrow, rw\downarrow, sw\uparrow, sw\downarrow\}$ and $C = \{cw\uparrow, cw\downarrow\}$) since in the composition, delay can occur in the inverter.

However, the two processes are identical if they operate in an environment that does not offer voltage changes on the input wires in situations where a voltage change on the output side can be expected. This can be expressed formally with the ∇_Z-operator of section 7.

2.3. THE C-ELEMENT.

The *Muller C-element* has two input wires and one output wire. After having received an input change on both input wires, it produces a change on the output wire. The graphical representation is shown in fig. 4.

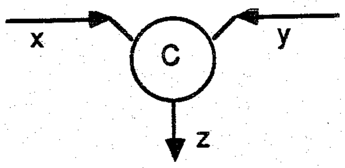


Fig. 4.

The process specification is as follows:

$$C^0_{xyz} = (rx\uparrow \parallel ry\uparrow) \cdot sz\uparrow \cdot C^1_{xyz}$$
$$C^1_{xyz} = (rx\downarrow \parallel ry\downarrow) \cdot sz\downarrow \cdot C^0_{xyz}.$$

We can incorporate an inverter with a C-element, graphically represented as in fig. 5. We call this element $C^0_{\underline{x}yz}$.

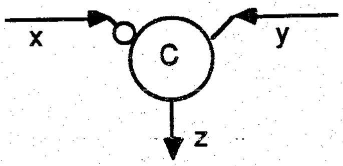


Fig. 5.

Such a C-element has the same specification as a regular C-element, with two differences: first, all arrows of actions on port x are reversed; second, when the voltages on wires x,y,z are all 0, it is in initial state $ry\uparrow \cdot sz\uparrow \cdot C^1_{\underline{x}yz}$. Observations similar to the ones at the end of 2.2, can be made in this case.

2.4. THE FORK.

The *fork*, graphically denoted by a • sign, has one input wire and two output wires. It passes the value of the input wire on to the output wires. In EBERGEN [E] essentially the following specification is given for the fork:

$$F^0_{xyz}=rx\uparrow\cdot(sy\uparrow \| sz\uparrow)\cdot F^1_{xyz}$$
$$F^1_{xyz}=rx\downarrow\cdot(sy\downarrow \| sz\downarrow)\cdot F^0_{xyz}$$

In this paper, we will not use forks of this type. This is because, for the correctness of a circuit, we have to make the assumption that the internal forks used to connect the components of the circuit are *isochronic* (see MARTIN [MA]), i.e. the delays in these forks are short enough, compared to the delays in other components, to assume that the outputs of an isochronic fork have the same value at any time. In the process algebra modeling an isochronic fork is just a *wire*:

$$W^0_{xy}=rx\uparrow\cdot sy\uparrow\cdot W^1_{xy}$$
$$W^1_{xy}=rx\downarrow\cdot sy\downarrow\cdot W^0_{xy}$$

At the output port of such a wire there are two processes reading. Thus we do not have binary but ternary communication. Up to now communication was defined by ($x \in P$):

$$sx\uparrow \mid rx\uparrow=cx\uparrow, sx\downarrow \mid rx\downarrow=cx\downarrow.$$

For ports $y \in P$ with ternary communication we want:

$$sy\uparrow \mid ry\uparrow \mid ry\uparrow=cy\uparrow,\ sy\downarrow \mid ry\downarrow \mid ry\downarrow=cy\downarrow.$$

We achieve this by defining the following binary communications:

$$sy\uparrow \mid ry\uparrow=sry\uparrow, sry\uparrow \mid ry\uparrow=cy\uparrow, ry\uparrow \mid ry\uparrow=rry\uparrow, sy\uparrow \mid rry\uparrow=cy\uparrow,$$

and similarly for the $\downarrow$-actions (of course $\mid$ is also commutative). The reader may think that defining synchronization in this way makes things more complicated than they actually are. Why not introduce synchronization as in trace theory (see e.g. VAN DE SNEPSCHEUT [S]) or TCSP (see e.g. HOARE [H]), where an arbitrary number of a-actions synchronize into just another a-action? Our motivation for doing it like this is that on the physical level an asymmetry exists between send and read actions. In our view it is important not to abstract too soon from this asymmetry. We will elaborate on this in the next section.

3. ACTIVE VERSUS PASSIVE.

In process algebra there is a symmetry between r- and s-actions. On the physical level such a symmetry is not present, and a distinction between *active* and *passive* actions can be made. A basic component can take the initiative to change the value on an output wire: the component performs an active s-action. If the value on the input wire of a component changes, this component is 'forced' to perform a passive r-action immediately. This means that, talking about circuits, there are problems with the physical interpretation of process expressions where a component wants to do a s-action, but cannot do so because the component on the other end of the wire is not prepared to do the corresponding r-action. As a part of the correctness proof of a circuit we therefore have to show that this *interference* does not occur.

A technique which makes a formal definition of the correctness criterion mentioned above possible is the *put mechanism* as presented in BERGSTRA [B]. Let us briefly review the put mechanism. Suppose that processes S and R are connected by a binary communication port x. The mechanism allows process S to perform an action $sx\Diamond$ ($\Diamond$ being either $\uparrow$ or $\downarrow$). This action can always be performed, regardless of whether or not R is able to receive the message. However, if R happens to be in a state that enables $rx\Diamond$, then $sx\Diamond$ and $rx\Diamond$ must synchronize into $cx\Diamond$. On the other hand, if $sx\Diamond$ is performed when $rx\Diamond$ is not enabled then no synchronization will occur.

Let $<_x$ be the partial ordering on the set of atomic actions defined by $sx\uparrow <_x cx\uparrow$ and $sx\downarrow <_x cx\downarrow$. Then the composition of S and R is given by:

$$\theta_{<x}\circ\partial_{\{rx\}}(S \,\|\, R).$$

The operator $\theta_{<x}$ gives $cx\uparrow$ priority over $sx\uparrow$, and $cx\downarrow$ over $sx\downarrow$, thus enforcing synchronization whenever possible. The operator θ was defined in BAETEN, BERGSTRA & KLOP [BBK1] by means of the system ACP_θ, which contains the axioms of ACP together with the following axioms in table 3. Here, the operator θ is parametrised by a given partial ordering $<$, and $\triangleleft$ is an auxiliary operator needed to give a finite axiomatisation; intuitively, $x\triangleleft y$ is x, but with all initial steps blocked, that are majorized by an initial step of y.

$\theta(a) = a$	$a\triangleleft b = a$	if not $a<b$
$\theta(xy) = \theta(x)\cdot\theta(y)$	$a\triangleleft b = \delta$	if $a<b$
$\theta(x + y) = \theta(x)\triangleleft y + \theta(y)\triangleleft x$	$x\triangleleft yz = x\triangleleft y$	
	$x\triangleleft(y + z) = (x\triangleleft y)\triangleleft z$	
	$xy\triangleleft z = (x\triangleleft z)y$	
	$(x + y)\triangleleft z = x\triangleleft z + y\triangleleft z$	

Table 3. ACP_θ.

EXAMPLE:

$$\begin{aligned}
&\theta_{<x}\circ\partial_{\{rx\}}(sx\uparrow \| a\cdot rx\uparrow) = \theta_{<x}\circ\partial_{\{rx\}}(sx\uparrow\cdot a\cdot rx\uparrow + a(sx\uparrow\cdot rx\uparrow + rx\uparrow\cdot sx\uparrow + cx\uparrow) + \delta) =\\
&\quad = \theta_{<x}(sx\uparrow\cdot a\cdot\delta + a(sx\uparrow\cdot rx\uparrow + \delta + cx\uparrow)) =\\
&\quad = \theta_{<x}(sx\uparrow\cdot a\cdot\delta)\triangleleft a(sx\uparrow\cdot rx\uparrow + cx\uparrow) + \theta_{<x}(a(sx\uparrow\cdot rx\uparrow + cx\uparrow))\triangleleft sx\uparrow\cdot a\cdot\delta =\\
&\quad = sx\uparrow\cdot a\cdot\delta\triangleleft a + a\cdot\theta_{<x}(sx\uparrow\cdot rx\uparrow + cx\uparrow)\triangleleft sx\uparrow =\\
&\quad = sx\uparrow\cdot a\cdot\delta + a\cdot(\theta_{<x}(sx\uparrow\cdot rx\uparrow)\triangleleft cx\uparrow + \theta_{<x}(cx\uparrow)\triangleleft sx\uparrow\cdot rx\uparrow) =\\
&\quad = sx\uparrow\cdot a\cdot\delta + a\cdot(sx\uparrow\cdot rx\uparrow\triangleleft cx\uparrow + cx\uparrow\triangleleft sx\uparrow\cdot rx\uparrow) =\\
&\quad = sx\uparrow\cdot a\cdot\delta + a\cdot(\delta + cx\uparrow) = sx\uparrow\cdot a\cdot\delta + a\cdot cx\uparrow.
\end{aligned}$$

In circuit specifications one can use the put mechanism to model communication between components. Showing that there is no danger of interference in a circuit then comes down to proving that on the internal ports no s-action can take place.

If there is ternary communication at port $y \in P$, the situation is a bit more complicated. We encapsulate actions $ry\Diamond$ and $rry\Diamond$ (read actions without a synchronizing send action), and impose on A the partial ordering $<_y$ given by $sy\uparrow <_y sry\uparrow <_y cy\uparrow$ and $sy\downarrow <_y sry\downarrow <_y cy\downarrow$ to express that we want to synchronize as much as possible. Absence of interference can now be proven by showing that on internal ports no s-actions or sr-actions can be performed.

Our definition of absence of interference is, in the setting of deterministic processes, equivalent to the definition in EBERGEN [E]. The advantage of our approach is however, that we can check the absence of interference in a neat way *while calculating the transition diagram of a circuit.* We do not need a separate round for it.

4. SEMANTICS.

In this paper we employ the axiom system ACP and extensions ACP_θ and ACP_τ which all correspond with bisimulation semantics. In BAETEN, BERGSTRA & KLOP [BBK2] it is shown that the operator θ is inconsistent with ready and failure semantics. Consequently the combination of θ and trace semantics (the semantics in which KALDEWAIJ [K] discussed the circuit we will consider here) is in general inconsistent. However, if one restricts attention to deterministic processes, then the priority operator can be added consistently to trace semantics. Essentially this is because on the domain of deterministic and deadlock-free processes bisimulation semantics and trace semantics coincide (see BAETEN & BERGSTRA [BB]).

Further, we use the *Recursive Definition Principle* (RDP) and the *Recursive Specification Principle* (RSP), that together say that every guarded recursive specification has a unique solution (see [BK3]). All specifications that occur in this paper are guarded.

5. SPECIFICATION.

In order to avoid interference, communication in electrical circuits often takes place following a handshaking protocol. Let a,b be an input wires of a given circuit and let $\bar{a}$,$\bar{b}$ be output wires. For a pair x,y of ports, the four-phase handshaking protocol demands that every pair of consecutive transitions on x and on y takes place on different wires. We are interested in a simple buffer (or semaphore) which follows the four-phase handshaking protocol for the pairs a,$\bar{a}$ and b,$\bar{b}$. This means that on one side of the circuit (the input side), it will perform an action $ra\uparrow$ followed by an action $s\bar{a}\uparrow$. Next, the voltages are changed back again ($ra\downarrow$ followed by $s\bar{a}\downarrow$). On the other side of the circuit, we get the actions $s\bar{b}\uparrow$, $rb\uparrow$, $s\bar{b}\downarrow$, $rb\downarrow$, in this order. Somehow, the buffer we want to construct must work in such a way, that the actions on the input side *cause* the actions on the output side. In the next section, we consider a particular implementation.

6. IMPLEMENTATION.

A proposal for an implementation, from KALDEWAIJ [K], is depicted below. Port names are displayed. Initial voltages are all 0.

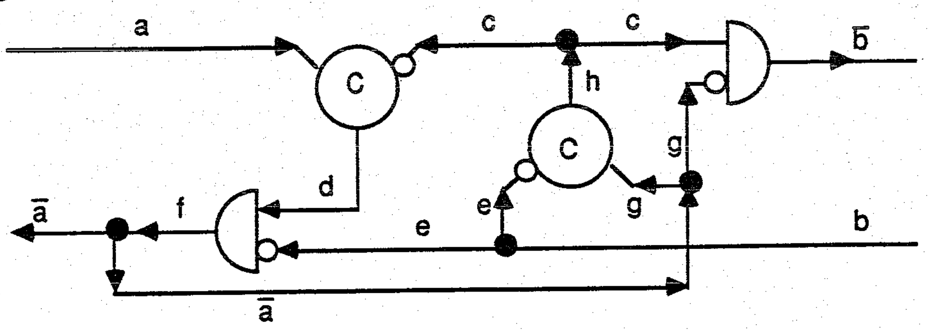


Fig. 6.

We now give a corresponding formal description. First we have a collection of port names:

$$P = \{a, \bar{a}, b, \bar{b}, c, d, e, f, g, h\}.$$

Of these, $\bar{a}$, c, e, g are ternary, the others (a, b, $\bar{b}$, d, f, h) are binary. Important sets of atomic actions are:

$$H = \{rx\uparrow, rx\downarrow, rrx\uparrow, rrx\downarrow : x = \bar{a},c,d,e,f,g,h\}$$

$$I = \{cx\uparrow, cx\downarrow : x = c,d,e,f,g,h\}$$

(actions $rrx\uparrow$, $rrx\downarrow$ matter only for ternary ports). Let $\theta_<$ be the priority operator based on the partial ordering on atomic actions given by:

$$sx\uparrow < srx\uparrow < cx\uparrow,\ sx\downarrow < srx\downarrow < cx\downarrow \qquad (x \in P)$$

(actions $srx\uparrow$, $srx\downarrow$ matter only for ternary ports).

Then the circuit of fig. 6 is given by:

$$IMPL = \theta_< \circ \partial_H(ra\uparrow \cdot sd\uparrow \cdot C^1_{\underline{c}ad} \parallel A^{10}_{\underline{e}df} \parallel W^0_{f\bar{a}} \parallel W^0_{\underline{a}g} \parallel W^0_{be} \parallel rg\uparrow \cdot sh\uparrow \cdot C^1_{\underline{e}gh} \parallel W^0_{hc} \parallel A^{10}_{gc\bar{b}})$$

Protocol verifications in BERGSTRA & KLOP [BK3] and VAANDRAGER [V] all have the form

$$\tau_J(\text{Implementation}) = \text{Specification}.$$

It seems that for circuits, this type of result cannot be obtained. Implementations can in general perform more sequences of actions at the input and output side than allowed by the specification. For instance, the process IMPL can start with a $rb\uparrow$-action. τ_I(IMPL) is only equal to SPEC in a *context* (environment) which behaves properly. For a correct behavior of the circuit it is necessary that actions from the environment are not offered too soon. Thus, we want to remove premature read-actions from the process IMPL. In order to formalize this, we need some trace theoretic notions from BAETEN & BERGSTRA [BB].

7. TRACE SETS.

7.1 DEFINITION. A^* is the set of finite sequences, or **traces**, of elements of A. The empty sequence is denoted by ε and sequence $\sigma * \rho$ is the concatenation of sequences σ and ρ. The domain $\mathbb{T}$ of **trace sets** is defined by: $\mathbb{T} = \{X \subseteq A^* : \sigma * \rho \in X \Rightarrow \sigma \in X\}$

(trace sets must be **prefix closed**). On $\mathbb{T}$, we define the operator $\partial/\partial a$ by ($Z \in \mathbb{T}$, $a \in A$):

$$\partial/\partial a(Z) = \{\sigma \in A^* : a * \sigma \in Z\}.$$

Further, we will have need of the **deletion operator** ε_I, that leaves out all elements of I in a trace. Here, I is a set of atomic actions ($I \subseteq A$). Note ε_I: $A^* \to A^*$.

7.2 DEFINITION. We define the **restriction of a process to a trace set.** If x is a process, and Z a trace set, then $\nabla_Z(x)$ is the result of disallowing every step in x that will result in a trace outside Z. Axioms for ∇_Z are in table 4, on the following page.

The restriction operator can be used to express that the process IMPL must behave in a certain way at the input and output side: all sequences of actions that do not conform to the four-phase handshaking protocol are disallowed. Thus, if in = A - $\{ra\uparrow, ra\downarrow, sr\bar{a}\uparrow, sr\bar{a}\downarrow\}$, out = A - $\{rb\uparrow, rb\downarrow, s\bar{b}\uparrow, s\bar{b}\downarrow\}$, we only allow traces in the set

$$Z = \{\sigma \in A^* : \varepsilon_{in}(\sigma) \text{ is a prefix of } (ra\uparrow sr\bar{a}\uparrow ra\downarrow sr\bar{a}\downarrow)^\omega,\ \varepsilon_{out}(\sigma) \text{ is a prefix of } (rb\uparrow s\bar{b}\uparrow rb\downarrow s\bar{b}\downarrow)^\omega\}.$$

$$\boxed{\begin{array}{ll}
\nabla_Z(\delta) = \delta & \\
\nabla_Z(\tau) = \tau & \\
\nabla_Z(\tau x) = \tau\cdot\nabla_Z(x) & \\
\nabla_Z(ax) = a\cdot\nabla_{\partial/\partial a(Z)}(x) & \text{if } a \in Z \\
\nabla_Z(ax) = \delta & \text{if } a \notin Z \\
\nabla_Z(x+y) = \nabla_Z(x) + \nabla_Z(y) &
\end{array}}$$

Table 4. Restriction operator.

We will show in the sequel that the process $\tau_I\circ\nabla_Z(\mathrm{IMPL})$ does behave like a buffer (the set I was defined in section 6). Notice that since port $\bar{a}$ is a ternary port, actions $sr\bar{a}$ occur here, and not actions $s\bar{a}$. This difference will become important in the sequel.

8. VERIFICATION.

Let the process SPEC be given by the following specification:

$$\mathrm{SPEC} = ra{\uparrow}\cdot\mathrm{SPEC}^*$$

$$\mathrm{SPEC}^* = sr\bar{a}{\uparrow}\cdot ra{\downarrow}\cdot sr\bar{a}{\downarrow}\cdot(s\bar{b}{\uparrow}\cdot rb{\uparrow}\cdot s\bar{b}{\downarrow}\cdot rb{\downarrow} \parallel ra{\uparrow})\cdot\mathrm{SPEC}^*.$$

Then the theorem below immediately shows that we have absence of interference.

8.1. THEOREM: $\tau_I\circ\nabla_Z(\mathrm{IMPL}) = \mathrm{SPEC}$.

PROOF: It will be useful to derive the complete transition diagram for process IMPL. This diagram is depicted in fig. 7. In this transition diagram, we have omitted all subtrees of edges, that are not allowed by the trace set Z, defined in section 7 above. Disallowed edges are made grey in the diagram.

For sake of completeness, we give the first part of the calculations that lead to the diagram:

$$\mathrm{IMPL} = \theta_{<}\circ\partial_H(ra{\uparrow}\cdot sd{\uparrow}\cdot C^1_{\underline{c}ad} \parallel A^{01}_{\underline{e}df} \parallel W^0_{f\bar{a}} \parallel W^0_{ag} \parallel W^0_{be} \parallel rg{\uparrow}\cdot sh{\uparrow}\cdot C^1_{\underline{e}gh} \parallel W^0_{hc} \parallel A^{10}_{\underline{g}c\bar{b}}) =$$

$$= ra{\uparrow}\cdot\theta_{<}\circ\partial_H(sd{\uparrow}\cdot C^1_{\underline{c}ad} \parallel A^{01}_{\underline{e}df} \parallel W^0_{f\bar{a}} \parallel W^0_{ag} \parallel W^0_{be} \parallel rg{\uparrow}\cdot sh{\uparrow}\cdot C^1_{\underline{e}gh} \parallel W^0_{hc} \parallel A^{10}_{\underline{g}c\bar{b}}) +$$

$$+ rb{\uparrow}\cdot\theta_{<}\circ\partial_H(ra{\uparrow}\cdot sd{\uparrow}\cdot C^1_{\underline{c}ad} \parallel A^{01}_{\underline{e}df} \parallel W^0_{f\bar{a}} \parallel W^0_{ag} \parallel se{\uparrow}\cdot W^1_{be} \parallel rg{\uparrow}\cdot sh{\uparrow}\cdot C^1_{\underline{e}gh} \parallel W^0_{hc} \parallel A^{10}_{\underline{g}c\bar{b}}) =$$

$$= ra{\uparrow}\cdot\big(cd{\uparrow}\cdot\theta_{<}\circ\partial_H(C^1_{\underline{c}ad} \parallel sf{\uparrow}\cdot A^{11}_{\underline{e}df} \parallel W^0_{f\bar{a}} \parallel W^0_{ag} \parallel W^0_{be} \parallel rg{\uparrow}\cdot sh{\uparrow}\cdot C^1_{\underline{e}gh} \parallel W^0_{hc} \parallel A^{10}_{\underline{g}c\bar{b}}) + rb{\uparrow}\cdot\ldots\big) +$$

$$+ rb{\uparrow}\cdot\ldots =$$

$$= ra{\uparrow}\cdot\big(cd{\uparrow}\cdot\big[cf{\uparrow}\cdot\theta_{<}\circ\partial_H(C^1_{\underline{c}ad} \parallel A^{11}_{\underline{e}df} \parallel s\bar{a}{\uparrow}\cdot W^1_{f\bar{a}} \parallel W^0_{ag} \parallel W^0_{be} \parallel rg{\uparrow}\cdot sh{\uparrow}\cdot C^1_{\underline{e}gh} \parallel W^0_{hc} \parallel A^{10}_{\underline{g}c\bar{b}}) +$$

$$+ ra{\downarrow}\cdot\ldots + rb{\uparrow}\cdot\ldots\big] + rb{\uparrow}\cdot\ldots\big) + rb{\uparrow}\cdot\ldots =$$

$$= ra{\uparrow}\cdot\big(cd{\uparrow}\cdot\big[cf{\uparrow}\cdot\big\{sr\bar{a}{\uparrow}\cdot\theta_{<}\circ\partial_H(C^1_{\underline{c}ad} \parallel A^{11}_{\underline{e}df} \parallel W^1_{f\bar{a}} \parallel sg{\uparrow}\cdot W^1_{ag} \parallel W^0_{be} \parallel rg{\uparrow}\cdot sh{\uparrow}\cdot C^1_{\underline{e}gh} \parallel W^0_{hc} \parallel A^{10}_{\underline{g}c\bar{b}}) +$$

$$+ ra{\downarrow}\cdot\ldots + rb{\uparrow}\cdot\ldots\big\} + ra{\downarrow}\cdot\ldots + rb{\uparrow}\cdot\ldots\big] + rb{\uparrow}\cdot\ldots\big) + rb{\uparrow}\cdot\ldots =$$

$$= ra{\uparrow}\cdot\big(cd{\uparrow}\cdot\big[cf{\uparrow}\cdot\big\{sr\bar{a}{\uparrow}\cdot\big(cg{\uparrow}\cdot\theta_{<}\circ\partial_H(C^1_{\underline{c}ad} \parallel A^{11}_{\underline{e}df} \parallel W^1_{f\bar{a}} \parallel W^1_{ag} \parallel W^0_{be} \parallel sh{\uparrow}\cdot C^1_{\underline{e}gh} \parallel W^0_{hc} \parallel A^{00}_{\underline{g}c\bar{b}}) + rb{\uparrow}\cdot..$$

$$+ ra{\downarrow}\cdot\theta_{<}\circ\partial_H(rc{\uparrow}\cdot sd{\downarrow}\cdot C^0_{\underline{c}ad} \parallel A^{11}_{\underline{e}df} \parallel W^1_{f\bar{a}} \parallel sg{\uparrow}\cdot W^1_{ag} \parallel W^0_{be} \parallel rg{\uparrow}\cdot sh{\uparrow}\cdot C^1_{\underline{e}gh} \parallel W^0_{hc} \parallel A^{10}_{\underline{g}c\bar{b}})\big)$$

$$+ra{\downarrow}\cdot\ldots+rb{\uparrow}\cdot\ldots\} + ra{\downarrow}\cdot\ldots + rb{\uparrow}\cdot\ldots] + rb{\uparrow}\cdot\ldots) + rb{\uparrow}\cdot\ldots = \ldots$$

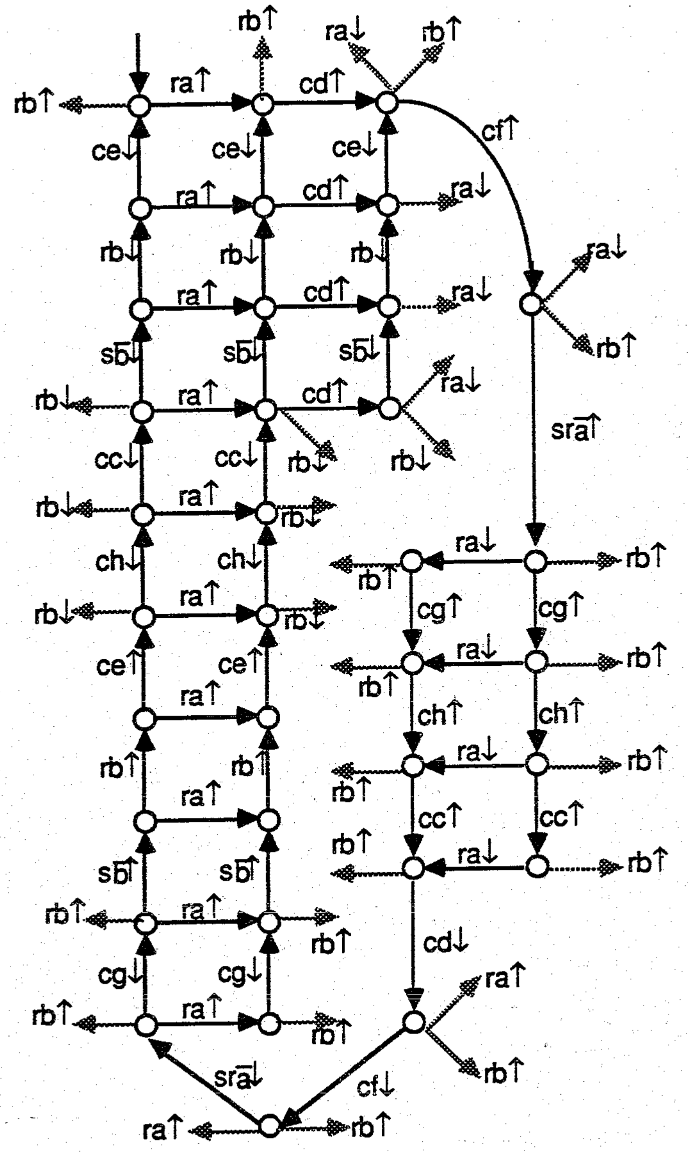


Fig. 7.

More straightforward calculations lead to the statement of the theorem:

$\tau_I\circ\nabla_Z(\text{IMPL}) =$

$$= ra{\uparrow}\cdot\tau\cdot\tau\cdot\tau_I\circ\nabla_Z\circ\theta_<\circ\partial_H(C^1_{\underline{c}ad} \parallel A^{11}_{\underline{e}df} \parallel s\bar{a}{\uparrow}\cdot W^1_{\bar{f}a} \parallel W^0_{ag} \parallel W^0_{\bar{b}e} \parallel rg{\uparrow}\cdot sh{\uparrow}\cdot C^1_{\underline{e}gh} \parallel W^0_{\bar{h}c} \parallel A^{10}_{\underline{g}c\bar{b}}) =$$

$$= ra{\uparrow}\cdot\tau_I\circ\nabla_Z\circ\theta_<\circ\partial_H(C^1_{\underline{c}ad} \parallel A^{11}_{\underline{e}df} \parallel s\bar{a}{\uparrow}\cdot W^1_{\bar{f}a} \parallel W^0_{ag} \parallel W^0_{\bar{b}e} \parallel rg{\uparrow}\cdot sh{\uparrow}\cdot C^1_{\underline{e}gh} \parallel W^0_{\bar{h}c} \parallel A^{10}_{\underline{g}c\bar{b}}),$$

and for the second equation:

$\tau_I\circ\nabla_Z\circ\theta_<\circ\partial_H(C^1_{\underline{c}ad} \parallel A^{11}_{\underline{e}df} \parallel s\bar{a}{\uparrow}\cdot W^1_{\bar{f}a} \parallel W^0_{ag} \parallel W^0_{\bar{b}e} \parallel rg{\uparrow}\cdot sh{\uparrow}\cdot C^1_{\underline{e}gh} \parallel W^0_{\bar{h}c} \parallel A^{10}_{\underline{g}c\bar{b}}) =$

$$= sr\bar{a}{\uparrow}\cdot[ra{\downarrow} \parallel \tau\cdot\tau\cdot\tau]\cdot\tau\cdot\tau\cdot sr\bar{a}{\downarrow}\cdot[ra{\uparrow}\cdot\tau \parallel \tau\cdot s\bar{b}{\uparrow}\cdot rb{\uparrow}\cdot\tau\cdot\tau\cdot\tau\cdot s\bar{b}{\downarrow}\cdot rb{\downarrow}\cdot\tau]\cdot$$

$$\cdot\tau_I\circ\nabla_Z\circ\theta_<\circ\partial_H(C^1_{\underline{c}ad}\,\|\,A^{11}_{\underline{e}df}\,\|\,s\bar{a}\!\uparrow\cdot W^1_{fa}\,\|\,W^0_{ag}\,\|\,W^0_{be}\,\|\,rg\!\uparrow\cdot sh\!\uparrow\cdot C^1_{\underline{e}gh}\,\|\,W^0_{hc}\,\|\,A^{10}_{gc\bar{b}})=$$

$$= sr\bar{a}\!\uparrow\cdot ra\!\downarrow\cdot sr\bar{a}\!\downarrow\cdot[ra\!\uparrow\,\|\,s\bar{b}\!\uparrow\cdot rb\!\uparrow\cdot s\bar{b}\!\downarrow\cdot rb\!\downarrow]\cdot$$
$$\cdot\tau_I\circ\nabla_Z\circ\theta_<\circ\partial_H(C^1_{\underline{c}ad}\,\|\,A^{11}_{\underline{e}df}\,\|\,s\bar{a}\!\uparrow\cdot W^1_{fa}\,\|\,W^0_{ag}\,\|\,W^0_{be}\,\|\,rg\!\uparrow\cdot sh\!\uparrow\cdot C^1_{\underline{e}gh}\,\|\,W^0_{hc}\,\|\,A^{10}_{gc\bar{b}})$$

(using the equation $\tau x \| y = \tau(x \| y)$). This finishes the proof of the theorem.

The theorem we just derived leads us to consider the question, in what sense this circuit really implements a buffer, in what sense do the actions on the input side really cause the actions on the output side. It is not the case that all input actions precede all output actions, as we can see from the theorem. But, we can reason as follows: of the four actions on the input side, the $sr\bar{a}\!\uparrow$ action is the 'real' input action, the acknowledgement that an input has taken place, the confirmation that input has been accepted. The $ra\!\uparrow$ action is the offering of the input, the down actions $ra\!\downarrow$ and $sr\bar{a}\!\downarrow$ are just re-initializations, to return to the original state of the circuit. On the other hand, we can consider the $s\bar{b}\!\uparrow$ action as the 'real' output action. The action $rb\!\uparrow$ is the acknowledgement of the environment that an output has been received, and the down actions are again re-initializations. (This choice of which actions constitute the 'real' input and output action coincides with observations of HOGERWOORD [HOO]).

This leads us to consider the following renaming function:

8.2. DEFINITION. On constants, define the following renaming function buf:

$$\mathrm{buf}(sr\bar{a}\!\uparrow) = \mathrm{input},\ \ \mathrm{buf}(s\bar{b}\!\uparrow) = \mathrm{output},$$
$$\mathrm{buf}(ra\!\uparrow) = \mathrm{buf}(ra\!\downarrow) = \mathrm{buf}(sr\bar{a}\!\downarrow) = \mathrm{buf}(rb\!\uparrow) = \mathrm{buf}(rb\!\downarrow) = \mathrm{buf}(s\bar{b}\!\downarrow) = \tau,$$

and buf leaves all other constants unchanged. Let ρ_{buf} be the operator that extends this function to all processes (see BERGSTRA & KLOP [BK3] for the notation; the renaming operators were introduced in BAETEN & BERGSTRA [BB]). Then we have the following theorem.

8.3 THEOREM: $\rho_{buf}(\mathrm{SPEC}) = \tau\cdot\mathrm{input}\cdot\mathrm{output}\cdot\rho_{buf}(\mathrm{SPEC})$.
(a correct specification for a one-bit buffer).

PROOF: $\rho_{buf}(\mathrm{SPEC}) = \tau\cdot\rho_{buf}(\mathrm{SPEC}^*)$, and

$$\rho_{buf}(\mathrm{SPEC}^*) = \mathrm{input}\cdot\tau\cdot\tau\cdot(\mathrm{output}\cdot\tau\cdot\tau\cdot\tau\,\|\,\tau)\cdot\rho_{buf}(\mathrm{SPEC}^*) =$$
$$= \mathrm{input}\cdot\mathrm{output}\cdot\tau\cdot\rho_{buf}(\mathrm{SPEC}^*) = \mathrm{input}\cdot\mathrm{output}\cdot\rho_{buf}(\mathrm{SPEC}).$$

Now we show how to use theorem 8.1 in a specific context. We take an environment that offers the actions concerned in the right order, e.g. we take the input process IN and the output process OUT given by the following guarded recursive specifications:

$$\mathrm{IN} = sa\!\uparrow\cdot r\bar{a}\!\uparrow\cdot sa\!\downarrow\cdot r\bar{a}\!\downarrow\cdot\mathrm{IN}$$
$$\mathrm{OUT} = r\bar{b}\!\uparrow\cdot sb\!\uparrow\cdot r\bar{b}\!\downarrow\cdot sb\!\downarrow\cdot\mathrm{OUT}.$$

8.4 THEOREM: Let $H^* = \{ra\!\uparrow, ra\!\downarrow, r\bar{a}\!\uparrow, r\bar{a}\!\downarrow, rr\bar{a}\!\uparrow, rr\bar{a}\!\downarrow, rb\!\uparrow, rb\!\downarrow, r\bar{b}\!\uparrow, r\bar{b}\!\downarrow\}$. Then process $\tau_I\circ\theta_<\circ\partial_{H^*}(\mathrm{IMPL}\,\|\,\mathrm{IN}\,\|\,\mathrm{OUT})$ is given by the following specification:

$$X = ca\!\uparrow\cdot Y,\ \ Y = c\bar{a}\!\uparrow\cdot ca\!\downarrow\cdot c\bar{a}\!\downarrow\cdot(c\bar{b}\!\uparrow\cdot cb\!\uparrow\cdot c\bar{b}\!\downarrow\cdot cb\!\downarrow\,\|\,ca\!\uparrow)\cdot Y.$$

PROOF: Define $H_a = \{ra\uparrow, ra\downarrow, r\bar{a}\uparrow, r\bar{a}\downarrow, rr\bar{a}\uparrow, rr\bar{a}\downarrow\}$. With the use of *conditional equations* similar to the ones described in BERGSTRA & KLOP [BK3], we can show

$$\tau_I \circ \theta_< \circ \partial_{H^*}(IMPL \,\|\, IN \,\|\, OUT) = \tau_I \circ \theta_< \circ \partial_{H^*}(\theta_< \circ \partial_{H_a}(IMPL \,\|\, IN) \,\|\, OUT).$$

Now consider the process $\theta_< \circ \partial_{H_a}(IMPL \,\|\, IN)$. This process has a transition diagram, similar to the diagram in fig. 7, with two differences: first, all grey $ra\Diamond$-edges are removed, and second, all the black a-edges ($ra\Diamond$, $sr\bar{a}\Diamond$) are changed to communication actions ($ca\Diamond$, $c\bar{a}\Diamond$). We do not calculate the subprocesses following grey $rb\Diamond$-actions of this process.

Then, we put this process in the context $\tau_I \circ \theta_< \circ \partial_{H^*}(... \,\|\, OUT)$. Then, all grey $rb\Diamond$-actions can be removed, and all black b-edges ($rb\Diamond$, $s\bar{b}\Diamond$) are changed to communication actions ($cb\Diamond$, $c\bar{b}\Diamond$).

The proof is finished if we rename all internal actions to τ-actions.

Theorem 8.1 is applicable in more contexts. As an example, we will put two copies of the circuit behind each other, and derive a specification for a two-bit buffer. Thus, consider the renaming functions am and $b\bar{m}$, where am renames ports a into m, and $b\bar{m}$ renames ports b into $\bar{m}$. To be more specific:

$$am(ra\uparrow) = rm\uparrow,\ am(s\bar{a}\downarrow) = s\bar{m}\downarrow,\ ...,\ b\bar{m}(cb\downarrow) = c\bar{m}\downarrow,\ b\bar{m}(r\bar{b}\uparrow) = rm\uparrow,\ ...,\ \text{etc.}$$

Now define

$$IMPL_{am} = \rho_{b\bar{m}}(IMPL),\ \ IMPL_{mb} = \rho_{am}(IMPL).$$

Thus, $IMPL_{am}$ is the buffer with input ports $a,\bar{a}$ and output ports $m,\bar{m}$, and $IMPL_{mb}$ is the buffer with input ports $m,\bar{m}$ and output ports $\bar{b}$,b.

Now define the sets $H_m = \{rm\uparrow, rm\downarrow, r\bar{m}\uparrow, r\bar{m}\downarrow, rr\bar{m}\uparrow, rr\bar{m}\downarrow\}$ and $Im = \{cm\uparrow, cm\downarrow, c\bar{m}\uparrow, c\bar{m}\downarrow\}$, and take the priority ordering as usual. Then we link the two buffers together, define

$$SPEC^2 = \tau_{I \cup I_m} \circ \nabla_Z \circ \theta_< \circ \partial_{H_m}(IMPL_{am} \,\|\, IMPL_{mb}).$$

Then we have the following theorem 8.5.

8.5 THEOREM: Process $SPEC^2$ is the process with the transition diagram depicted in fig. 8. The two nodes marked with a 1 should be identified, as well as the two nodes marked with a 2. (It is easy to give a recursive specification for this process, but that would not be very illuminating.)

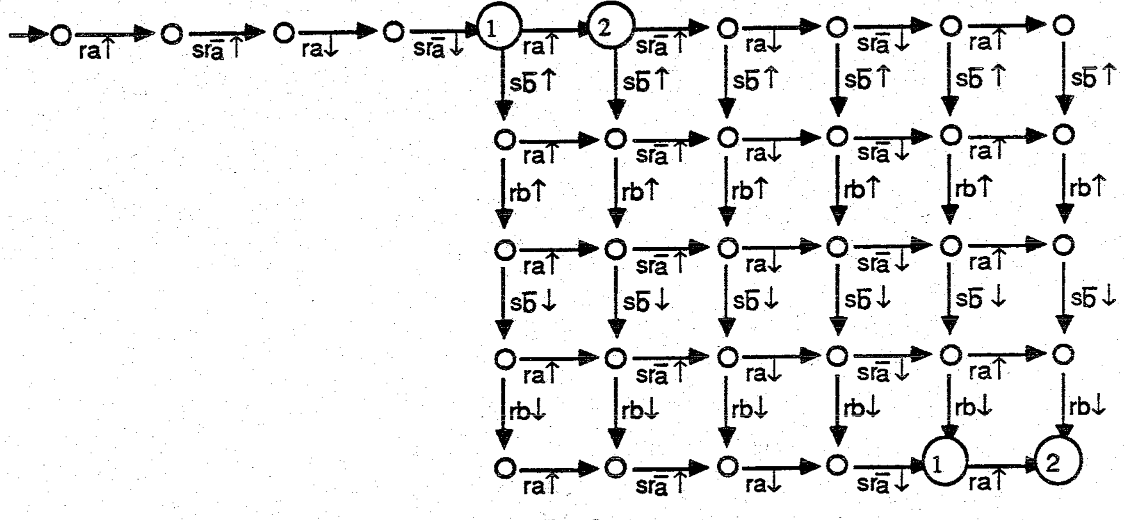


Fig. 8.

PROOF: Of course, we can obtain transition diagrams for IMPL_{am} and IMPL_{mb} from fig. 7 by just renaming actions. We must show that all grey actions can be omitted. First, IMPL_{mb} can undertake no action until it has received an input. Its first input cannot come along b for that is forbidden by ∇_Z. Therefore, the first action of IMPL_{mb} is the $rm\uparrow$ communicating with $sm\uparrow$ of IMPL_{am}. Then, IMPL_{am} has already performed 12 or 13 actions, and all grey edges before this point can be disregarded. Now, IMPL_{am} must wait for an answer of IMPL_{mb} (or perform an $ra\uparrow$-action, and then wait). This comes when IMPL_{mb} executes $sr\overline{m}\uparrow$, and we see that we can omit its grey edges before this point.

Continuing in this fashion, we see we can omit all grey edges, so there is no interference, and what remains is a great quantity of not very interesting calculations.

We get a easier to understand specification for the process SPEC^2 if we apply the renaming function buf of 8.2. Then we obtain the following theorem.

8.6 THEOREM: The process $\rho_{buf}(\mathsf{SPEC}^2)$ satisfies the following recursive specification:

$$X_0 = \tau \cdot \mathsf{input} \cdot X_1$$
$$X_1 = (\mathsf{input} \,\|\, \mathsf{output}) \cdot X_1.$$

This is indeed a correct specification for a two-bit buffer.

PROOF: As 8.3.

9. DELAY-INSENSITIVITY.

Central in the theory of circuit design is the notion of **delay-insensitivity**. Intuitively, a circuit is delay-insensitive if its behavior does not depend on delays in wires and switching elements. It seems that there is no general agreement in the literature about the precise definition of the concept. Formal definitions can be found in [U] and [E]. A minimum requirement for a circuit to be delay-insensitive is that it is free of interference. Another property which is usually associated with delay-insensitivity is the **Foam Rubber Wrapper Principle** of [MFR]. This postulate says that the external behavior of a circuit is invariant if the output ports are extended by wires: the circuit is thought of as being wrapped in foam rubber of which the boundaries are flexible and possibly changing (see figure 9).

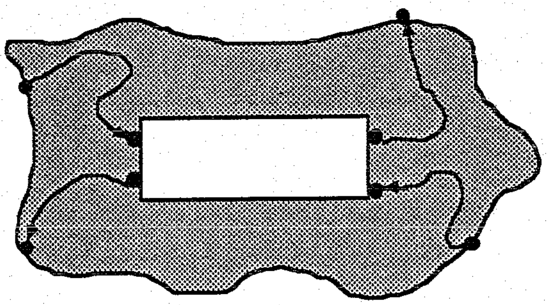


Fig. 9.

The behavior of the circuit we are dealing with in this paper does not satisfy the Foam Rubber Wrapper Principle. In order to demonstrate this, we extend port $\bar{a}$ by a wire. A new port $\overline{m}$ is introduced together with a renaming function transforming actions at $\bar{a}$ into actions at $\overline{m}$:

$$pr(s\bar{a}) = s\overline{m},\ pr(sr\bar{a}) = sr\overline{m}.$$

If <' is the partial ordering given by $s\overline{m} <' sr\overline{m} <' c\overline{m}$ and $H' = \{r\overline{m}, rr\overline{m}\}$, then the extended circuit is defined by: $\quad IMPL' = \theta_{<'}\circ\partial_{H'}(W_{\overline{m}a} \| \rho_{pr}(IMPL))$.

Analogously to theorem 8.1, we can derive the following specification for the process $SPEC' = \tau_I\circ\nabla_Z(IMPL')$:

$$SPEC' = ra\uparrow\cdot SPEC''$$
$$SPEC'' = sr\bar{a}\uparrow\cdot ra\downarrow\cdot(s\bar{b}\uparrow\cdot rb\uparrow\cdot s\bar{b}\downarrow\cdot rb\downarrow \| sr\bar{a}\downarrow\cdot ra\uparrow)\cdot SPEC''.$$

Thus, processes SPEC' and SPEC have different external behavior. However, SPEC' is still interference-free and is a correct implementation of a one-bit buffer according to definition 8.2. Thus, while the circuit of this paper does not satisfy the Foam Rubber Wrapper Principle in the strict sense, functional behavior *is* preserved, when we extend the external ports with wires.

In general, it is not possible to give an interference free implementation of the specification SPEC which satisfies the Foam Rubber Wrapper Principle. The argument for this is simple: the circuit has to perform send-actions at ports $\bar{a}$ and $\bar{b}$ right after each other. After having sent a message at $\bar{a}$, the circuit does not know whether or not the environment has received this message, due to the unbounded (although finite) delay in the wire. After waiting some time the circuit has to send a message at $\bar{b}$, which may arrive too early. Current research ([E]) suggests that if we have a specification with an interference free implementation composed of basic elements like wires, forks and C-elements, the output ports which are not tapped satisfy the Foam Rubber Wrapper Principle. As we have seen this is not true in general for specifications with tapped output wires. Now we can observe that the fact that in specification SPEC $sr\bar{a}$-actions occur instead of $s\bar{a}$-actions already tells us that port A presumably does not satisfy the Foam Rubber Wrapper Principle. The wire at port $\bar{a}$ is tapped and therefore the boundary of the circuit is not flexible there.

10. CONCLUSIONS.

For the hierarchical design and verification of a complex circuit it is crucial that we can reason about the circuit in terms of the external behavior (specification) of smaller components of the circuit. We want to abstract from the implementation details of these components. We have seen that it is unwise to abstract from the distinction between send, read and communication actions, as that makes it possible to see immediately whether or not our implementation is interference-free, and also makes it possible to identify ports that do not satisfy the Foam Rubber Wrapper Principle.

We used the priority operator θ to describe the put mechanism of BERGSTRA [B]. This makes it possible to distinguish active and passive actions, and so we can deal with interference. However, the drawback is that we cannot use the abstraction operator τ_I inside the scope of the θ operator, as this could destroy the interference information. We can illustrate this with a simple example:

$\tau_{\{i\}}\circ\theta_{s<c}\circ\partial_{\{r\}}(s \| i\cdot r) = s\delta + \tau c$, which displays interference, while

$\theta_{s<c}\circ\tau_{\{i\}}\circ\partial_{\{r\}}(s \| i\cdot r) = \tau c$ is interference-free.

Thus, we cannot abstract from the internal actions in the scope of the priority operator, which makes modular verification more difficult. Recent research suggests that it may be possible to affect part of the abstraction inside the scope of the priority operator, using a constant which has less laws than τ, and do the remainder of the abstraction outside.

REFERENCES.

[BB] J.C.M. BAETEN & J.A. BERGSTRA, *Global renaming operators in concrete process algebra,* report CS-R8521, Centre for Math. and Comp. Sci., Amsterdam 1985. *Revised version:* report P8709, Programming Research Group, University of Amsterdam 1987. To appear in Inf. & Comp.

[BBK1] J.C.M. BAETEN, J.A. BERGSTRA & J.W. KLOP, *Syntax and defining equations for an interrupt mechanism in process algebra,* Fund. Inf. IX (2), pp. 127 - 168, 1986.

[BBK2] J.C.M. BAETEN, J.A. BERGSTRA & J.W. KLOP, *Ready trace semantics for concrete process algebra with the priority operator,* British Comp. Journal 30 (6), pp. 498 - 506, 1987.

[B] J.A. BERGSTRA, *Put and get, primitives for synchronous unreliable message passing,* report LGPS 3, Dept. of Philosophy, State University of Utrecht, 1985.

[BK1] J.A. BERGSTRA & J.W. KLOP, *Process algebra for synchronous communication,* Information & Control 60 (1/3), pp. 109-137, 1984.

[BK2] J.A. BERGSTRA & J.W. KLOP, *Algebra of communicating systems with abstraction,* Theor. Comp. Sci. 37 (1), pp. 77-121, 1985.

[BK3] J.A. BERGSTRA & J.W. KLOP, *Process algebra: specification and verification in bisimulation semantics,* in: Math. & Comp. Sci. II (M.Hazewinkel, J.K.Lenstra & L.G.L.T.Meertens, eds.), CWI Monographs 4, pp. 61 - 94, North-Holland, Amsterdam 1986.

[E] J.C. EBERGEN, *Translating programs into delay-insensitive circuits,* Ph.D.Thesis, Technical University Eindhoven, 1987.

[H] C.A.R. HOARE, *Communicating sequential processes,* Prentice Hall 1985.

[HOO] R. HOOGERWOORD, *Some reflections on the implementation of trace structures,* Computing Science Notes 86/03, Dept. of Math. & Comp. Sci., Eindhoven University of Technology 1986.

[K] A. KALDEWAIJ, *The translation of processes into circuits,* in: Proc. PARLE Conf. (Vol. I), (J.W. de Bakker, A.J.Nijman & P.C.Treleaven, eds.), pp. 195 - 212, Springer LNCS 258, 1987.

[MA] A.J. MARTIN, *Compiling communicating processes into delay-insensitive VLSI circuits,* Distr. Comp. 1, pp. 226 - 234, 1986.

[MI] R. MILNER, *A calculus of communicating systems,* Springer LNCS 92, 1980.

[MFR] C.E. MOLNAR, T.-P. FANG & F.U. ROSENBERGER, *Synthesis of delay-insensitive modules,* in: Proc. Chapel Hill Conf. on VLSI (H. Fuchs, ed.), pp. 67-86, Computer Science Press, 1985.

[R] M. REM, *Partially ordered computations, with applications to VLSI design,* Proc. Found. of Comp. Sci. IV.2 (J.W. de Bakker & J. van Leeuwen, eds.), MC Tract 159, pp. 1 - 44, Mathematical Centre, Amsterdam 1983.

[S] J.L.A. VAN DE SNEPSCHEUT, *Trace theory and VLSI design,* Springer LNCS 200, 1985.

[U] J.T. UDDING, *A formal model for defining and classifying delay-insensitive circuits and systems,* Distr. Comp. 1, pp. 197 - 204, 1986.

[V] F.W. VAANDRAGER, *Verification of two communication protocols by means of process algebra*, report CS-R8608, Centre for Math. and Comp. Sci., Amsterdam 1986.