Acknowledgement

The authors are very grateful to Dr. Hubert Garavel and Dr. Laurent Mounier for their valuable assistance, leading to improved versions of this and related papers.
(2) For any \( w_1 \in L(S\lot) \) and \( w' \in L(CT\lot) \), with \( w' \models w_1 \), let

\[
q_0(S\lot)[w_1 > a \quad \text{and} \quad q_0(CT\lot)[w' > b
\]

Then we set \( (a, b) \in R \).

Evidently, domain \( R \) contains all the states of \( S\lot \setminus CT\lot \) and range \( R \) contains all the states of \( S\lot \).

Let now \( (a, b) \in R \) and assume that \( w_2 \) is applicable to \( a \). Then \( w_1;w_2 \in L(S\lot) \), and

\[
q_0(S\lot)[w_1;w_2 > c, \quad \text{where} \quad a[w_2 > c.
\]

(5) implies that there exists \( w'' \), such that \( w'' \models w_2 \) and \( w';w'' \in L(CT\lot) \). Hence,

\[
q_0(CT\lot)[w';w'' > d, \quad \text{where} \quad b[w'' > d.
\]

But \( (w';w'') \models w_1;w_2 \). By definition of \( R \), we have \( (c, d) \in R \).

It follows that \( R \) is indeed an observational equivalence, i.e., Condition C1 holds.
of CT.lot, and there exists a word \( w \in L(S \cdot \text{lot}) \) and a word \( w' \in L(CT \cdot \text{lot}) \), such that \( w' \overset{\cdot}{i} = w \) (\( w' \overset{\cdot}{i} \) is obtained from \( w' \) by omitting all instances of ‘\( i \)’ in \( w' \)'), and
\[ q_0(S \cdot \text{lot})[w > a , q_0(CT \cdot \text{lot})[w' > b. \]

Condition C1 implies that there exists a relation R satisfying the requirements of the relevant observation equivalence. Clearly,
\[ q_0(S \cdot \text{lot} | \ | CT \cdot \text{lot}) = (q_0(S \cdot \text{lot}), q_0(CT \cdot \text{lot})). \]

Notice that \( (q_0(S \cdot \text{lot}), q_0(CT \cdot \text{lot})) \) is a state of \( S \cdot \text{lot} | \ | CT \cdot \text{lot} \), in view of the above considerations, with \( w = w' = \lambda \).

Furthermore, \( q_0(S \cdot \text{lot} | \ | CT \cdot \text{lot}) R q_0(S \cdot \text{lot}) \).

Let now \( w_1 \) and \( w' \) be as in (5). Also, let \( q_0(S \cdot \text{lot})[w_1 > a \) and \( q_0(CT \cdot \text{lot})[w' > b \).

Then, \( (a, b) \) is a state of \( S \cdot \text{lot} | \ | CT \cdot \text{lot} \). In view of the relevant observation equivalence, we have \( (a, b) R a \). Furthermore, assume \( a[w_2 > c \). In view of the above observation equivalence, there exists a state \( (a', b') \) in \( S \cdot \text{lot} | \ | CT \cdot \text{lot} \), such that \( a[w_2 > a' \) in \( S \cdot \text{lot} \), \( b[w' > b' \) in \( CT \cdot \text{lot} \), and \( w' \overset{\cdot}{i} = w_2 \). It follows that \( w';w'' \) is applicable to \( q_0(CT \cdot \text{lot}) \), i.e., \( w';w'' \in L(CT \cdot \text{lot}) \).

Thus, (5) is established.

A similar reasoning shows that C1 => (3).

Proposition 3.2

(5) => Condition C1.

Proof

In order to establish Condition C1, it is necessary and sufficient to establish a relation R between the states of \( S \cdot \text{lot} | \ | CT \cdot \text{lot} \) and the states of \( S \cdot \text{lot} \), which satisfies the requirements of observational equivalence. This relation R may be defined as follows.

(1) \( (q_0(S \cdot \text{lot}), q_0(CT \cdot \text{lot})) = q_0(S \cdot \text{lot} | \ | CT \cdot \text{lot}) R q_0(S \cdot \text{lot}) \)
Proof of Propositions 3.1 and 3.2  

In this Appendix we prove Propositions 3.1 and 3.2, formulated in Chapter 3.

We start by recalling the basic concepts involved.

Let $S$ be a specification, and $CT$ a circuit transition system. Below we repeat item (5)
of the Definition of $CT | = S$ (Definition 1.4.2).

(5) Assume $w_1; w_2 \in L(S)$, $w' \in L(CT)$, where $w' \text{intCT} = w_1$.

Then there exists $w''$, such that $w'' \text{intCT} = w_2$, and $w'; w'' \in L(CT)$.

We denote by $S \text{lot}$ a LOTOS-process, observation-equivalent to $S$. $CT\text{int}$ is obtained
from $CT$ by replacing all internal symbols by 'i'.

$CT \text{lot}$ is a LOTOS-process, obs.-equiv. to $CT\text{int}$. 

Condition C1

$S \text{lot} || CT \text{lot}$ is obs.-equiv. to $S \text{lot}$.

In the following, $S \text{lot}$, $CT \text{lot}$, and $S \text{lot} || CT \text{lot}$ refer to the corresponding LOTOS
processes, as well as to the corresponding LTSs (automata).

Proposition 3.1

Condition C1 => Requirements (3) and (5) of Definition 1.4.2

Proof

We first prove C1 => (5).

$w_1; w_2 \in L(S) \Rightarrow w_1; w_2 \in L(S \text{lot})$, (in view of the observational equivalence between $S$
and $S \text{lot}$).

Now consider the LTS (automaton) representing $S \text{lot} || CT \text{lot}$.

Its states may be viewed as ordered pairs $(a, b)$, where $a$ is a state of $S \text{lot}$, $b$ is a state
A Guide to LOTOS Literature

Appendix A

A. WWW/FTP Sites

[utwente]/ http://wwwtios.cs.utwente.nl/lotos/
[uottawa1]/ http://www.csi.uottawa.ca/~lotos/
[uottawa2]/ ftp://lotos.csi.uottawa.ca/pub/Lotos/
[upm]/ http://www.dit.upm.es/~lotos/
[stir1]/ http://www.cs.stir.ac.uk/~kjt/research/
[stir2]/ ftp://ftp.cs.stir.ac.uk/pub/staff/kjt/research/pubs/
[inria]/ http://www.inrialpes.fr/vasy/cadp.html

B. Tutorials

in: [stir2]/lotos_users.ps.gz

in: [uottawa2]/Papers/index.html

C. Applications to Hardware

References

[Async] The Asynchronous Logic Home Page, at
http://www.cs.man.ac.uk/amulet/async/index.html

[BS94] J.A.Brzozowski and C-J.H.Seger, Asynchronous Circuits,

[Dill89] D.L.Dill, Trace Theory for Automatic Hierarchical Verification
of Speed-Independent Circuits,

[KG97] R.Kol and R.Ginosar, Future Processors will be Asynchronous,

[McM95] K.L.McMillan, A Technique of State Space Search Based on Unfolding,

[RCP95] O.Roig, J.Cortadella, and E.Pastor,
Verification of asynchronous circuits by BDD-based model checking

[Roi97] O.Roig, Formal Verification and Testing of Asynch. Circuits,
Doct.Informatica Thesis, Universitat Politecnica de Catalunya,

[Sut89] I.E.Sutherland, Micropipelines,

[Yoe87] M.Yoeli, Specification and Verification of Asynchronous Circuits,
Using Marked Graphs, in: Concurrency and Nets,
Using CADP, one verifies that Condition C2 is satisfied by X4.IMP.aut, the LTS-representation of X4.IMP.

### 3.2.3 The No-Undesirable Output Condition

Applying CADP, Condition C3 is verified.

This concludes the proof of VE.X4.

### 3.3 Conclusion

In additional publications we demonstrate the applicability of the above verification approach to various, more complex examples.
We postulate that a given specification is deadlock-free. In this case Condition C3 is evidently implied by Condition C1.

In the verification example that follows, the considerations of this section will be illustrated.

### 3.2 Verification Example VE.X4

In this section we deal with the following verification example:

\[ \text{VE.X4: } X4.\text{IMP } |\Rightarrow X4.\text{SP} \]

Here, \( X4.\text{SP} \) specifies a 4-input XOR-gate (\( \text{XOR}4 \)), and \( X4.\text{IMP} \) represents a possible implementation, containing three 2-input XOR-gates.

In the following we use the simplified LOTOS-based notation introduced in Chapter 2.

#### The Specification X4.SP

\[ X4.\text{SP}= a;z;X4.\text{SP} \parallel b;z;X4.\text{SP} \parallel c;z;X4.\text{SP} \parallel d;z;X4.\text{SP} \]

Note that in\( \text{XOR}4=\{a,b,c,d\} \) and out\( \text{XOR}4=\{z\} \).

#### The Implementation X4.IMP

\[ X4.\text{IMP}= \\
\langle \text{XOR}[a,b,r] \parallel \text{XOR}[c,d,s] \parallel [r,s] \rangle \text{XOR}[r,s,z]\{r,s\} \]

### 3.2.1 Verification of Condition C1

Using the formal LOTOS-representations of \( X4.\text{SP} \) and \( X4.\text{IMP} \), which are easily derived from the above simplified notation, and applying the toolbox CADP, one verifies that Condition C1 is indeed satisfied.

### 3.2.2 The No-Livelock Requirement
Proposition 3.2

(5) of Definition 1.4.2 implies Condition C1.

Thus, Requirement (3) is a consequence of Requirement (5).

For a proof of the above two propositions see Appendix B.

3.1.2 The No-Livelock Condition

An automaton, such as CT.aut, is said to be livelock-free, iff it does not contain a cycle, all edges of which belong to intCT. In this case, CT.aut evidently satisfies (6) of the Definition 1.4.2. The above no-livelock condition is satisfied in many practical examples.

Consequently, we introduce

Condition C2

CT.aut is livelock-free.

Using CADP, we can verify this requirement by means of the following command:

    aldebaran -live CT.aut

3.1.3 The No-Undesirable-Output Condition

Here we refer to (4) of the Definition 1.4.2. This condition is satisfied if the following requirement is met:

Condition C3

S.lot || CT.lot contains no deadlock (in the sense of LOTOS).
Such an interconnection is modelled in LOTOS by means of the partial synchronisation operator $|p|$. In case the interconnection is declared as internal port of the network, the LOTOS hiding operator is applied.

**Chapter 3 - LOTOS-Based Verification**

### 3.1 The Basic Approach

In this chapter we relate the concepts of Chapter 1 to the LOTOS-oriented discussions of Chapter 2. We will frequently refer to the details of the definition of $CT|=S$ (Definition 1.4.2).

#### 3.1.1 The Major Requirement

Let $CT$ be a circuit transition system. We denote by $CT\text{\scriptsize int}$ the LTS obtained from $CT$ by replacing all its internal symbols by \textquoteleft i\textquoteright. Let $CT\text{\scriptsize lot}$ denote a LOTOS-process, which is observation-equivalent to $CT\text{\scriptsize int}$.

Given a specification $S$, we denote by $S\text{\scriptsize lot}$ a LOTOS-process, observation-equivalent to $S$.

We now formulate a set of conditions which ensure that $CT|=S$ holds. They can be checked using CADP.

The checking of (1) and (2) of the Definition of $CT|=S$ is evidently trivial.

**Condition C1**

$S\text{\scriptsize lot} \mid \mid CT\text{\scriptsize lot}$ is observation-equivalent to $S\text{\scriptsize lot}$.

**Proposition 3.1**

Condition C1 implies (3) and (5) of Definition 1.4.2.
After applying the relevant CADP commands (see above) we get:

```
file XOR.omin

des (0,3,2)
    (0,B,1)
    (0,A,1)
    (1,Z,0)
```

Note that the numbers (0,3,2) appearing in the first line above have the following meaning:

0 = initial state, 3 = # of arcs, 2 = # of nodes.

Furthermore, note that for the above example, the graph representation coincides with \( f_X \) of the comp.a representation (Section 2.3.1).

### 2.4 Modular Networks

A modular network is obtained by suitably interconnecting a finite number of modules. The modules of such a network must adhere to the following restriction.

If \( p \) is an input of any module, there may be at most one module with \( p \) as output.

In such a case the output \( p \) of this particular module is connected to every input \( p \) occurring in any other module. Such an interconnection \( p \) may be declared either as an output port of the network, or as an internal port.
We are concerned with a XOR-gate with binary inputs A and B, and binary output Z.

The gate is **stable** iff \((A \text{ xor } B) = Z\).

In the descriptions below we deal with the dynamic behavior of the gate.

### 2.3.1 LTS Representation

**Definition 2.3.1**

The LTS XOR.a is defined as the 5-tuple

\[
\text{XOR.a} = (\text{inX, outX, QX, fX, q0X}),
\]

where

- \(\text{inX} = \{a, b\}\),
- \(\text{outX} = \{z\}\),
- \(\text{QX} = \{0, 1\}\),
- \(\text{q0X} = 0\), and
- \(\text{fX} = \{(0,a,1),(0,b,1),(1,z,0)\}\).

Note that the labels a, b, z denote ports ("gates"), as well as events occurring at these ports.

Note further, that this component adheres to the "fundamental-mode" restriction.

### 2.3.2 The Simplified LOTOS-based Notation

Let XOR.b be the representation in this notation. Then XOR.b = \(\text{[(a;$[b;$)>z;$]*}\)

Alternatively, XOR.b can be represented recursively as follows:

\[
\text{XOR.b} = a;z;\text{XOR.b} \text{[} b;z;\text{XOR.b}\]
\]

### 2.3.3 The LOTOS Specification

The file XOR.lotos

specification XOR[A,B,Z]: noexit behaviour

\[
\text{QXOR[A,B,Z]}
\]

where

process QXOR[A,B,Z]: noexit :=
A component comp.a is defined by a 5-tuple \((\text{in}, \text{out}, Q, f, q_0)\), where

(a) \(\text{in}\) and \(\text{out}\) are finite sets of input ports and output ports, respectively.

(b) \(Q\) is a finite set of states.

(c) \(f\) is a partial function from \(Q \times (\text{in} \cup \text{out})\) into \(Q\).

(d) \(q_0 \in Q\) is the initial state.

Notice that this definition is a special case of Definition 1.3.1, where \(\text{intCT} = \emptyset\).

### 2.2.3 A Simplified LOTOS-Based Notation

In this notation we use the major LOTOS operators, namely \([\cdot]\), the three parallel operators, and \(\gg\). For convenience, we introduce the following modified notation:

1. \(\$ = \text{exit}\)

2. If \(P\) is a process which terminates successfully, we denote by \(P^*\) the process \(P^* = P \gg P^*\).

3. If \(B\) is a behaviour expression and \(H\) is a set of events, participating in \(B\), then \(B \setminus H\) is the behaviour obtained from \(B\) by hiding the events in \(H\).

### 2.2.4 LOTOS Specifications

For the purpose of computer-aided verification, we use proper LOTOS specifications. The corresponding file will be denoted comp.lotos. By applying the CADP command `caesar -aldebaran comp.lotos`, a file comp.aut will be generated, providing a graph-based version of the original specification. The command `aldebaran -omin comp.aut > comp.omin` provides a reduced version (comp.omin), observation-equivalent to comp.aut. CADP also provides a facility to check observation equivalence between two comp.out graphs.

We illustrate the above representation methods by means of the following example.

### 2.3 The Two-Input XOR-Gate
The purpose of this and the following chapter is to illustrate the applicability of LOTOS and its related toolbox CADP (see Appendix A) to an automated verification of asynchronous circuits.

2.2 The Concept of Module

We assume asynchronous circuits, composed of basic components ("modules"), which are more complex than simple gates. The correct behavior of such modules is to be ensured by proper circuit design (at the transistor level), rather than by logic design.

Such a module may be in a stable condition (i.e., no output will change, as long as no input occurs), or otherwise be in an unstable condition. We assume that each module adheres to the following "fundamental mode restriction":

(*) In a stable condition an input may be applied; in an unstable condition no input may be applied, but an output must occur.

2.2.1 Representation of Components

In this section we discuss methods of representing the dynamic behaviour of basic components (modules).

For such basic components we establish three methods of representation:

(a) By means of labeled transition systems (LTS);
(b) By using a simplified LOTOS-based notation;
(c) By means of proper LOTOS specifications (process descriptions).

2.2.2 LTS-Representation

Our LTS-representation of (the dynamic behaviour of) components is summarized in the following definition.

Definition 2.2.1
Requirement (5) becomes redundant.

Informally, the above Definition 1.4.2 may be motivated as follows.

(a) the input and output symbols of CT are assumed to coincide with inS and outS, respectively (Requirements (1) and (2)).

(b) we expect the realization to be capable of producing for any signal sequence specified by $L(S)$ a sequence which, after deletion of its internal symbols, will be equal to the above signal sequence. CT may even be capable of performing more than required by $S$ (Requirement (3), cf. [Dill89]).

(c) the realization may not produce any "undesirable" outputs (Requirement (4)).

(d) Requirement (5) is rather evident; as mentioned above, this Requirement is redundant, if CT is deterministic.

On the other hand, one easily verifies that Requirement (5) implies Requirement (3) (see also Section 3.1.1).

(e) a word of the realization which reduces to a word in $L(S)$, cannot be continued by an infinite sequence of internal signals (Requirement (6)).

1.5 Related Work

For related discussions of the topic of this Chapter, see [BS94]/Chapter 11, as well as [Dill89].

Chapter 2 - LOTOS-based Behavior Descriptions

2.1 Introduction

In Chapter 1 we introduced the basic concepts of realization, with respect to asynchronous circuits.

In this and the following chapter we model a circuit and its specification by a CTS, which will be represented by a suitable, LOTOS-based behaviour description.
1.4 Specifications and their Realizations

**Definition 1.4.1**

A Specification $S$ is a CTS, where $\text{intCT} = \emptyset$. We set $S = (\text{inS}, \text{outS}, \text{QS}, fS, q0S)$. The language $L(S)$ is defined similarly to the above definition of $L(\text{CT})$.

The following definition is intended to formalize the intuitive concept "CT is a realization of $S$".

**Definition 1.4.2**

Let $S = (\text{inS}, \text{outS}, \text{QS}, fS, q0S)$ be a specification and CT a circuit transition system, $CT = (\text{inCT}, \text{outCT}, \text{intCT}, \text{QCT}, fCT, q0CT)$.

We say that CT is a realization of $S$ (notation: $CT \models S$), iff the following conditions are satisfied.

Note: We use ‘;’ to denote concatenation.

1. $\text{inCT} = \text{inS}$
2. $\text{outCT} = \text{outS}$
3. $L(S) \subseteq L(\text{CT}) \setminus \text{intCT}$
4. Assume $w \in L(S)$, $z \in \text{outS}$, $w';z \in L(\text{CT})$ and $w' \setminus \text{intCT} = w$.
   Then $w;z \in L(S)$.
5. Assume $w1,w2 \in L(S)$, and there exists $w' \in L(\text{CT})$, such that $w' \setminus \text{intCT} = w1$.
   Then there exists $w''$, such that $w''\setminus \text{intCT} = w2$ and $w';w'' \in L(\text{CT})$.
6. Let $w \in L(S)$, $w' \in L(\text{CT})$, and $w' \setminus \text{intCT} = w$.
   Then there exists a positive integer $k$ such that for any word $w'' \in (\text{intCT})^*$,
   $w';w'' \in L(\text{CT})$ implies $\text{length}(w'') < k$.

Clearly, if $CT\setminus$ is deterministic, Requirement (5) is implied by Requirement (3), i.e.,
next. A CTS also includes information as to the restrictions imposed on the environment. These restrictions are intended to ensure the delay-insensitivity of the circuit, as illustrated in Section 1.2.1.

**Definition 1.3.1**

A *Circuit Transition System* CT is a 6-tuple

\[ CT = (\text{in}_{CT}, \text{out}_{CT}, \text{int}_{CT}, Q_{CT}, f_{CT}, q_{0CT}). \]

\( \text{in}_{CT} \) is a finite set of *input symbols*. An input symbol represents an input node of the relevant circuit, as well as a change of logic level (i.e., from 0 to 1, or from 1 to 0) at the corresponding node. Similarly, \( \text{out}_{CT} \) and \( \text{int}_{CT} \) are finite sets of *output symbols* and *internal symbols*, respectively. Output symbols and internal symbols represent corresponding circuit nodes, as well as their level changes. The sets \( \text{in}_{CT}, \text{out}_{CT}, \) and \( \text{int}_{CT} \) are mutually disjoint.

We set \( \text{in}_{CT} \cup \text{out}_{CT} \cup \text{int}_{CT} = a_{CT}. \)

\( Q_{CT} \) is a finite set of *states*.

\( f_{CT} \), the *next-state function*, is a partial function from \( Q_{CT} \times a_{CT} \) into \( Q_{CT} \).

\( q_{0CT} \in Q_{CT} \) is the *initial state* of CT.

We postulate that all states in \( Q_{CT} \) are reachable from the initial state.

Let CT be a circuit transition system, as defined above. CT may be viewed as an acceptor automaton with \( a_{CT} \) as its alphabet. We consider every state of \( Q_{CT} \) to be an accepting state, and denote by \( L(C) \) the corresponding language.

Let \( w \in L(C) \). We denote by \( w \backslash \text{int}_{CT} \) the restriction of \( w \) to the alphabet \( \text{in}_{CT} \cup \text{out}_{CT} \), and set \( L(C) \backslash \text{int}_{CT} = \{ w \backslash \text{int}_{CT} \mid w \in L(C) \} \).

We denote by \( C \backslash \) the automaton obtained from CT by replacing its internal symbols by \( \varepsilon \). Evidently, \( C \backslash \) may be non-deterministic, although CT has been defined as a deterministic system.
behavior of asynchronous circuits is quite similar to the way, processes/agents are described, e.g., in CSP, CCS, and LOTOS.

Note that we use the same symbol to denote a rising transition as well as a falling transition at a given node.

1.2.1 An Illustrative Example

To illustrate our approach to dynamic behavior descriptions, we consider a simple combinational gate, namely an Inverter. Assume such an inverter has a binary-level input A, and a binary-level output Z. Notice that this inverter is stable (i.e., no output change will occur, as long as the input does not change) iff \( A \neq Z \).

We impose the following restriction on the behavior of the environment: an input change may occur only if the inverter is stable. This restriction is to ensure the proper, predictable behavior of the gate, independently of its propagation delay. If we were to admit an input change, while the gate is still unstable, the outcome would be unpredictable (an output pulse may or may not occur, depending on the actual timing conditions). Since we are interested in delay-insensitive behavior, such an environment restriction is well justified.

Thus we describe the dynamic behavior of this inverter by a finite automaton Inv.a with state set \( \{q_0,q_1\} \). Here state \( q_0 \) of Inv.a represents the two stable states of the inverter, and state \( q_1 \) of Inv.a represents the two unstable states (i.e., \( A=Z \)). The alphabet of Inv.a is \( \{a,z\} \). E.g., the symbol ‘a’ denotes a level change at input port A, namely both an up-transition (from level 0 to level 1) as well as a down-transition. The state transitions of Inv.a are \( (q_0,a,q_1) \) and \( (q_1,z,q_0) \). The initial state is \( q_0 \).

1.3 Circuit Transition Systems (CTS)

In this section we model the behavior of an asynchronous circuit by means of a modified version of an LTS (Labeled Transition System), called CTS, to be defined
In Chapter 1 we discuss our basic approach to the modelling of asynchronous circuits, their specification and realization. In particular, we introduce a mathematically precise formalism, establishing the relationship between specification and realization.

In Chapter 2 we introduce LOTOS-based descriptions of basic asynchronous components (modules) and of modular networks.

In Chapter 3 we discuss and illustrate the LOTOS-based automated verification of asynchronous, modular networks.

Chapter 1 - Asynchronous Circuits: Basic Concepts of Realization

1.1 Introduction

In this chapter we discuss our approach to the modelling of asynchronous circuits, their specifications, and their realizations. In general, we describe circuit realizations, as well as their specifications, by means of their "dynamic behavior".

1.2 Dynamic Behavior

The conventional approach to the behavioral description of digital circuits is based on observing the values of inputs and outputs at suitably chosen, discrete points in time. In a synchronous circuit, for example, the frequency of these observations is in accordance with the clock rate. In contrast to this "static" approach to behavioral descriptions, the "dynamic" approach applied in this paper, is based on observing "external events" of the circuit, i.e. changes of the value of an input or output. A dynamic description of the behavior of a digital system consists of listing all admissible sequences of external events, subject to restrictions imposed on the environment. We are interested in the order in time, in which the events occur, and not in the actual time intervals involved. The dynamic viewpoint is particularly applicable, and is indeed widely used, with respect to asynchronous circuits. This approach to the
Chapter 0 - Introduction

This paper deals with the formal verification of asynchronous circuits. In particular, it demonstrates the applicability of the high-level specification language LOTOS to the verification tasks in question. Furthermore, we use the LOTOS-oriented toolbox CADP for the purpose of automating our verification approach.

An extensive literature is presently available dealing with asynchronous circuits. See, e.g., [BS94], which provides an extensive list of references, as well as the web-site [Async].

Asynchronous (clock-free) systems have important advantages over (globally clocked) synchronous systems, particularly in view of the present trend toward high-speed and high-density technologies. For more details, see [Sut89],[Async],[BS94],[KG97].

A large amount of the presently available literature deals with (correct-by-construction) synthesis. The literature on formal verification is rather limited. We mention [Yoe87],[Dill89],[McM95],[RCP95],[Roi97].

We find that using LOTOS and CADP provides us with tools which compete favorably with other published approaches to the verification problem under discussion, particularly with respect to modular, observationally non-deterministic, asynchronous circuits.

For information on LOTOS see Appendix A. Information on CADP is available at the web-site Appendix A/[inria]/.

LOTOS and CADP have proven powerful tools for the high-level specification of communication protocols, and their verification. In [FL93] and [TS94] the applicability of LOTOS to the formal specification of hardware systems was demonstrated. The application of LOTOS and CADP to the formal specification and verification of asynchronous hardware, presented in this paper, appears to be new.
Abstract:

The paper introduces a precise formulation of the relationship between specification and realization of asynchronous circuits.

It demonstrates the suitability of the high-level specification language LOTOS, and its related toolbox CADP to the automated verification of modular, asynchronous circuits.