NET-BASED SYNTHESIS OF DELAY-INSENSITIVE CIRCUITS

by

Michael Yoeli

Technical Report #609

February 1990
NET-BASED SYNTHESIS OF DELAY-INSENSITIVE CIRCUITS

Michael Yoeli
Computer Science Department
Technion - Israel Institute of Technology
Haifa 32 000, Israel

ABSTRACT

This paper is an introduction to a novel formal theory of delay-insensitive circuits, their verification and synthesis. We consider asynchronous circuits, obtained by suitably interconnecting basic components ("modules"). Such a circuit is "delay-insensitive" if its correct behavior is independent of the delays of its modules and its connecting wires.

The novel features of our approach are the following:

(1) We develop a suitable formal delay and race model of modular circuits;
(2) We define in a new way the concept "implementation satisfies specification";
(3) We use (a suitable algebraic representation of) Petri nets for both the high-level specification as well as the synthesis of delay-insensitive circuits. Our synthesis method is "direct", i.e. we do not need to transform the given specification net into a finite-state machine.

The synthesis method derived in this paper is restricted to certain classes of Petri nets. However, research is in progress to expand the applicability of our design approach.
1. INTRODUCTION

This paper is intended as an introduction to a novel theory of delay-insensitive circuits, their verification and their synthesis. We are concerned with asynchronous circuits, i.e. circuits which are not controlled by a global clock. Furthermore, we deal with "modular" circuits, i.e. circuits obtained by the suitable interconnection of "basic" components ("modules"), to be defined later on (see Section 3). Such a modular circuit is "delay-insensitive", if its correct behavior is independent of the delays of its modules and its connecting wires. All these concepts will be made precise in the sequel (see Sections 3-6).

Important research contributions towards a formal theory and a design methodology for delay-insensitive circuits have been published, particularly [BKR85], [Re85], [Sn85], [Ma86], [Ch87], [Eb87], [RMCF88], [MBM89]. Our approach differs from the preceding publications in a number of ways:

(1) We develop a delay and race model for modular circuits, based on a suitable modification of the "General Multiple Winner" race model for asynchronous gate networks introduced in [BY79].

(2) We define in a rather novel way the concept "implementation satisfies specification". This definition expresses formally the idea that an implementation may be more powerful than required by its specification, but on the other hand should not produce "undesirable" outputs.

(3) We use Petri nets for both the (high-level) specification as well as the synthesis of delay-insensitive circuits. In [MFR85], [Ch87], [MBM89] Petri nets are also used, but for circuit specifications only. Alternative approaches to the use of Petri nets for both the specification and the implementation of asynchronous circuits can be found in [PD73], [BKR85], [Yo87a].

The synthesis method derived in this paper (Section 8) is restricted to certain classes
of Petri nets, particularly marked graphs. However, research is in progress to further expand the applicability of our design approach (see Section 9). In Section 2 we summarize the advantages of delay-insensitive circuits over their synchronous counterpart.

This paper uses material, mostly in a suitably revised form, from [YE83], [Yo87b], [Re88], [RY88], [YR89].

2. ASYNCHRONOUS VS. SYNCHRONOUS CIRCUITS

Although synchronous circuits are used nearly exclusively in present-day VLSI practice, the type of asynchronous circuits usually referred to as "delay-insensitive" has important advantages over their synchronous counterparts (see e.g. [Re85], [Ma86], [Eb87], [RMCF88]). The designer of synchronous circuits has to cope with the difficulty of distributing a global clock over a large chip area. His design is sensitive to the length of connecting wires and therefore to the final layout of his design. The designer has to take into careful account the feasible worst-case delays of basic components and their interconnections. Thus timing analysis at the system level becomes a difficult problem (cf. e.g. [WS89]).

On the other hand the correct operation of delay-insensitive circuits is independent of the actual delay distribution. Hence the delay-insensitive approach facilitates a structured design methodology with respect to both logic and timing, whereas in synchronous designs timing considerations are necessarily always global. Furthermore, the speed of operation of a synchronous circuit is determined by the worst-case delay distribution, whereas the speed of any asynchronous circuit depends only on actual delays. Thus delay-insensitive circuits are on the average faster than their synchronous counterpart.
3. MODULES

3.1 Basic Concepts

The modules (basic components) we use in this paper have one or two binary inputs and one binary output. They are both combinational (gates), as well as sequential. We assume that their global state is completely specified by their input/output state. In particular, if their input/output state is stable, they are also stable internally. This can be achieved by a suitable circuit design within an "equipotential region" (cf.[Se80]).

The behavior of a module can be completely specified by listing its stable input/output (i/o) states. The behavior of a module is determined as follows:

(a) if its i/o state is stable, its output will not change as long as its inputs do not change.
(b) if its i/o state is unstable, its output will change, provided the inputs remain unchanged.
(c) if its i/o state is unstable, but becomes stable due to an input change before its output has changed, this transient unstable state will have no influence on the behavior of the module.

However, Rule (c) is only an approximation to the actual behavior of a module: if a number of transient unstable states follow each other with short intervals between them, the behavior of the module becomes rather unpredictable. Consequently, we shall impose suitable restrictions on the environment of a circuit, in order to avoid the situation described in Rule (c).

Formally we describe a two-input module $M$ with inputs $A$ and $B$, and output $Z$ as follows: $M[A,B;Z] 1= SP$, where $SP$ (Stability Predicate) is a predicate on $A,B,Z$ (viewed as boolean variables), specifying the stable i/o states of $M$. Similarly, a one-input module is specified by $M[A;Z] 1= SP$.

In this paper we shall use the following modules:

- $WIRE[A;Z] 1= Z = A$
INV[A;Z] \models Z = \neg A

AND[A,B;Z] \models Z = A \land B

OR, XOR, NAND, NOR are defined similarly.

The sequential (Müller) C-Element, which plays an important role in the design of delay-insensitive circuits (cf. [Se80]) is specified by

CE[A,B;Z] \models (A = Z \land B = Z) \lor (B = \neg A).

3.2 Behavior of Modules

Consider e.g. a two-input module, with inputs A and B, and output Z. Assume that this module is activated by its environment in some way. An external observer will notice a sequence of input and output changes. We denote by e.g. A+ (A-) a change of A from 0 to 1 (from 1 to 0). For example, if AND[A,B;Z] is started in the state 000 (i.e. A=0, B=0, Z=0), the following sequence of states and events (input and output changes) is feasible:

```
000, A+, 100, B+, 110, Z+, 111, A-, 011, Z-, 010, B-, 000.
```

The corresponding observable event sequence is:

```
<A+, B+, Z+, A-, Z-, B->
```

Given a module M and its initial state q, we define the 'event behavior' of (M,q) as the set of all feasible event sequences of M, when started in state q. Note that we are only concerned with the sequence of events, and not their actual timing relation. Hence our considerations will be independent of the actual values of the module delays involved. Our concept of event behavior corresponds to the way (communicating sequential) processes are described in [Ho85].

Frequently the behavioral description of a circuit may be simplified by representing both transitions of a binary input or output, say X+ and X-, by a single symbol, say x. We shall refer to these symbols as (input or output) 'signals'. Clearly, an event sequence such as the above sequence, can be reconstructed from the corresponding signal
sequence \(<a,b,z,a,z,b>\), provided the initial state 000 is given. The 'signal behavior' of a module is defined similarly to its 'event behavior'.

So far we have restricted our considerations to the case where events occur one at a time. Later on we shall deal with the general case, where two or more events may occur simultaneously. We shall also extend our behavioral descriptions to networks consisting of suitably interconnected modules (see Subsection 4.2).

4. MODULAR NETWORKS

4.1 Network Diagrams

We now consider networks consisting of suitably interconnected modules. We first represent a network by a 'network diagram'. Later on we introduce a suitable algebraic notation, similar to the one proposed by Dill [Di89] (cf. also [Yo87b]).

A network diagram is a finite directed graph with the following node types:

(a) External nodes (terminals) shown as small circles. An external node is either an 'input terminal' with indegree 0 or an 'output terminal' with indegree 1.

(b) Internal nodes shown as black dots. They are hidden from an outside observer. Their indegree is 1. Both external and internal nodes are labeled by upper-case letters, with subscripts if necessary. We use A,B,C for input terminals, Y,Z for output terminals, and X for internal nodes.

(c) Module nodes shown as squares with labels, identifying the type of the module, written inside the square. They represent the corresponding modules and have indegree 1 or 2 and outdegree 1. Exactly one node of every edge in the diagram must be a module node.

An example of a modular network (NWK) is shown in Figure 4.1.

Two networks NWK1 and NWK2 are 'compatible' iff the following conditions are met:

(a) any shared label is a terminal label in both networks;

(b) any shared label may not be an output terminal label in both networks.
In this case we define their 'composition', denoted NWK1 ⊕ NWK2, as follows:
NWK = NWK1 ⊕ NWK2 is the interconnection of NWK1 and NWK2, obtained by identifying terminals with equal labels. If two input terminals are identified, the outcome is again an input terminal of NWK. If an input terminal and an output terminal are identified, then the outcome is an output terminal of NWK. Clearly, the ⊕ operator is associative and commutative.

We also consider the case, where some or all of the outcomes of identifying an input terminal with an output terminal become internal nodes of NWK, hidden from an outside observer. To cover this case, we introduce the "hiding" operator \, using the notation NWK = (NWK1 ⊕ NWK2) \ {X.1,...,X.k} , where the X.i's are the hidden nodes.

The network NWK of Figure 4.1 may be described as follows:
NWK = (CE[A,B;Y] ⊕ INV[C,X] ⊕ CE[X,Y;Z]) \ {X}

A state of a network NWK is an assignment of binary values (0,1) to the external and internal nodes of NWK. A 'circuit' CCT is a pair (NWK,q) , where q is a state of NWK. Let CCT1 = (NWK1,q1) and CCT2 = (NWK2,q2). CCT1 and CCT2 are 'compatible' iff this is the case with respect to NWK1 and NWK2, and furthermore q1 and q2 are "compatible", i.e. terminals having equal labels also have equal values. The operator ⊕ is extended to circuits in the evident sense.

Two networks NWK1 and NWK2 are 'strongly compatible', iff they are compatible, and furthermore the following additional condition is met:
(c) any shared label may not be an input terminal label in both networks.

In this case we use the notation NWK1 # NWK2 to denote their composition, with the hiding operator applied to all nodes which have been obtained by identifying terminals. Thus terminals with equal labels in NWK1 and NWK2 become internal nodes of the combined network NWK1 # NWK2. The definition of '#' is again extended to circuits, similarly to the extension of '⊕'.
4.2 Behavior of Circuits

Let $CCT = (NWK,q)$ be a circuit, where $NWK$ is represented by a network diagram. The labels of the external and internal nodes represent binary variables. We use the corresponding lower-case letters to denote the associated signals, i.e. relevant up- and down-transitions. Thus we associate with $NWK$ an input signal alphabet $inNWK$, an output signal alphabet $outNWK$, and an internal signal alphabet $intNWK$.

Given a state $q$ of $NWK$, we define the local state of each module node in the obvious way. The node representing the output of a module (which is either an internal node or an output node of $NWK$) is 'stable' iff the corresponding module node is stable.

A state of $NWK$ is 'stable' iff all its internal nodes and output terminals are stable.

Let $q$ be a stable state of $NWK$ and let $s$ be a nonempty subset of $inNWK$. Then $s$ is applicable to state $q$. We write $q[s>q'$, where $q'$ is the state obtained by applying $s$ to state $q$. Let now $q'$ be an unstable state of $NWK$, and let $T$ denote any nonempty set of nodes, each of which is either an input node or an unstable node. Then $t$, the set of signals which corresponds to the node set $T$, is applicable to state $q'$ and we use again the notation $q'[t>q''$. Using the above model of $NWK$, we now proceed to define $msb(NWK,q_0)$, i.e. the "multiple-signal behavior" of $(NWK,q_0)$, for a given initial state $q_0$ of $NWK$. Let $q_0=q_0,q_1,...,q_n$ be a sequence of states of $NWK$ and $t_1,...,t_n$ a sequence of signal sets, such that $q_i[t_{i+1}>q_{i+1}$ for $0 \leq i < n$. Then

$q_0,t_1,q_1,...,t_n,q_n$

is a 'state/signal sequence' of $(NWK,q_0)$. Let $w = s_1,...,s_m$, where $m \leq n$, be the signal-set sequence obtained from $t_1,...,t_n$ by omitting all internal node signals in the obvious way (we again use the notation $q_0[w>q_n$). The sequence $w$ thus obtained is a 'multiple-signal sequence' of $(NWK,q_0)$. $msb(NWK,q_0)$ is defined as the set of all such multiple-signal sequences of $(NWK,q_0)$, including the empty sequence $\lambda$.

To illustrate, we return to the network $NWK$ of Figure 4.1. We shall represent a
state of NWK by means of the bit vector <A,B,C,X,Y,Z>. Let q0 = 001000 be the initial state of NWK. The following are examples of state/signal sequences of (NWK,q0).

seq1: 001000, {a,b,c}, 110000, {x,y}, 110110, {z}, 110111;
seq2: 001000, {a,b}, 111000, {y}, 111010, {a}, 011010, {c}, 010010, {x}, 010110, {z}, 010111.

Thus <(a,b,c),(y),(z)> and <(a,b),(y),(a),(c),(z)> are multiple-signal sequences of CCT = (NWK,q0).

Note that the above circuit analysis is based on the Rules (a),(b),(c), specifying the behavior of a module (see Subsection 3.1). It could be argued that this analysis tends to disregard hazards which may occur (cf. [Di89]). However, such a hazard becomes detectable, by replacing a given module by its "wire extension". This point will be further discussed in Subsection 5.4.

Our approach to circuit behavior is based on input and output changes ("dynamic behavior") rather than on input and output levels ("static behavior"). The dynamic behavior approach is used extensively in connection with the application of trace theory to VLSI design [Re85], [Sn85], [Eb87], [Di89]. However, our approach also takes into account the possibility of a number of signals occurring simultaneously, i.e. it is capable of representing "true concurrency", whereas the trace-theoretical approach replaces "true concurrency" by "non-deterministic interleaving". Multiple signals are also admissible in the GMW model of [BY79], as well as in G.J.Milne's CIRCAL Calculus (see e.g. [DM87]).

5. CIRCUIT SPECIFICATIONS

We assume circuits to be specified by "directed processes", to be defined next.

5.1 Directed Processes

Given a finite alphabet A, let ext(A) be the alphabet consisting of all nonempty
Let $P_1 = ((a,b), (z), (abz))$. Then

$CCT_1 = (\text{AND}(A,B;\text{Z}),000)$ is evidently an implementation of $P_1$, since $000,a,100,b,110,z,111$ is a state/signal sequence of $CCT_1$.

However, $P_1$ has also a simpler implementation, e.g.

$CCT_2 = ((\text{WIRE}(A;X) \parallel \text{WIRE}(B;\text{Z}) \setminus \{X\},0000)$, where a state of $CCT_2$ is represented by the bit vector $<A,B,X,Z>$. Indeed,

$0000,a,1000,x,1010,b,1110,z,1111$

is a state/signal sequence of $CCT_2$, and consequently $abz$ belongs to $\text{msb}(CCT_2)$. Clearly, the above Requirements (1), (3), and (4) are also met.

On the other hand, let $P_2 = ((a,b), (z), (a,b,abz))$. $CCT_1$ is still an implementation of $P_2$, but $CCT_2$ is not, although $(a,b,abz)$ is a subset of $\text{msb}(CCT_2)$. However, Requirement (3) is not satisfied: $<b>$ is a sequence belonging to both $b(P_2)$ and $\text{msb}(CCT_2)$, but $bz$ is in $\text{msb}(CCT_2)$, but $b$ is not in $b(P_2)$.

5.4 Delay-Insensitive Implementations

We now turn to delay-insensitivity. As mentioned in the introduction we are interested in the design of delay-insensitive circuits, i.e. asynchronous circuits which perform correctly, independently of the particular delay distribution of the circuit. The delays in question may be due to wires connecting the circuit to the environment, or wires providing the connections between modules, or due to the internal (propagation) delays of the modules themselves. We shall now formulate the relevant concept more formally.

Following [Eb87] we define delay-insensitivity by means of a modified version of the "Foam Rubber Wrapper" principle [MFR85]. For this purpose we define "wire extensions" of the modules used in this paper. E.g. the wire extension of $\text{OR}(A,B;Z)$ becomes:

$\text{WOR}(A,B;A.1,B.1,Z.1,Z) = (\text{OR}(A.1,B.1,Z.1,Z) \parallel \text{WIRE}(A;A.1) \parallel \text{WIRE}(B;B.1) \parallel \text{WIRE}(Z.1,Z)) \setminus \{A.1,B.1,Z.1\}$
The wire extensions of the other modules, except WIRE, are defined similarly. The wire extension of WIRE[A;Z] coincides with WIRE[A;Z]. For a given circuit CCT let WCCT be the circuit obtained by replacing each module by its wire extension. The given state is extended in such a way, as to have all the added WIRE modules in a stable state. We say that CCT is a 'di-implementation' of P (denoted: CCT di-sat P) iff WCCT sat P.

By replacing CCT by WCCT we are in a position to detect hazards which were not evident in CCT. For example, let CCT = (OR[A,B;Z],000) and P = ((a,b),(z),pref<a,z,(a,b),b,z>, where, for any sequence s, pref(s) denotes the set of prefixes of s, including s and the empty sequence. CCT is an implementation of P, in view of the following state/signal sequence:

600,a,100,z,011,{a,b},011,b,001,z,000.

However, CCT is not a di-implementation of P. Namely, consider the following state/signal sequence of WCCT (we represent a state of WCCT by the bit vector <A,B,A,1,B,1,Z,1,Z>):

000000,a,100000,a,1,010000,z,1,010100,z,101011,(a,b)
011011,a,1,010011,(b,1,z,1),010101,z,010100

Thus, <a,z,(a,b),z> is a sequence in msb(WCCT). In view of Requirement (3) of Subsection 5.2, WCCT is not an implementation of P, i.e. 'CCT di-sat P' does not hold.

6. PETRI NETS

In this section we discuss the representation of directed processes by means of labeled Petri nets. Our approach to Petri nets differs from the conventional one (cf.[Pe81]) in two aspects:

(1) Rather than relying on the usual graphical representations of Petri nets, we introduce a suitable algebraic notation.
(2) We consider multiple-firing sequences, i.e. we assume that a number of transitions may fire simultaneously.

6.1 Basic Concepts

This and the following subsection are based on [YE83].

A 'Petri net' is a triple $N = (P, T, V)$, where

1. $P$ and $T$ are finite sets of 'places' and 'transitions', respectively;
2. $P$ and $T$ are disjoint;
3. $V$ is a function $V : (P \times T) \cup (T \times P) \rightarrow \{0, 1\}$.

A 'marking' of $N$ is a function from $P$ into the set $\{0, 1, 2, \ldots\}$ of non-negative integers. A 'marked net' is a pair $S = (N, M)$, where $N$ is a net, and $M$ is a marking of $N$.

Let $X$ be a nonempty set of transitions of $S = (N, M)$. $X$ is 'concurrently firable' in $S$, iff the following holds for every place $p$:

$$M(p) \geq \sum\{ V(p,t) \mid t \in X \},$$

where $\sum\{ x_1 \ldots x_n \}$ denotes $x_1 \oplus \ldots \oplus x_n$, for any binary, associative operation $\oplus$.

If $X$ is firable in $S = (N, M)$, then the 'firing' of $X$ yields a new marking $M'$, where $M'$ satisfies the following condition for every place $p$ of $S$:

$$M'(p) = M(p) - \sum\{ V(p,t) \mid t \in X \} + \sum\{ V(t,p) \mid t \in X \}.$$

We use the notation $M[X>M']$, assuming $N$ to be fixed.

Consider now a marked net $S = (N, M)$, where $N = (P, T, V)$. Let $X.1, \ldots, X.n$ be nonempty subsets of $T$, and $M.1, \ldots, M.n$ markings of $N$ such that

$$M[X.1>M.1, M.1[X.2>M.2, \ldots, M.n-1[X.n>M.n].$$

We then call $X.1, \ldots, X.n$ a 'multiple-firing sequence' of $S$, and write $M[X.1,\ldots,X.n>M.n]$. We denote by $pL(S)$ the set of all multiple-firing sequences of $S$, including the empty (multiple-firing) sequence $\Lambda$. Thus $pL(S)$ is a prefix-closed language (the "parallel" language of $S$) over the alphabet $\text{ext}(T)$.
6.2 Labeled Nets

A 'labeled net' is a triple \( H = (S, A, f) \), where \( S = (P, T, V, M) \) is a marked net, \( A \) is a finite alphabet, and \( f \) is a mapping

\[
f : T \to A \cup \{ \lambda \},
\]

where \( \lambda \) denotes the empty sequence. \( H \) is '\( \lambda \)-free' iff \( f(T) \subseteq A \).

The mapping \( f \) may be interpreted as a labeling of the transitions of \( S \), and will be referred to as the "labeling" function of \( H \). Similarly, \( A \) is the "labeling" alphabet of \( H \).

We now extend the labeling function \( f \) to sequences in \( pL(S) \) as follows. For any nonempty subset \( X \) of \( T \), let

\[
f(X) = \begin{cases} \{ f(t) \mid t \in X \} = \{ \lambda \} & \text{if } \{ f(t) \mid t \in X \} = \{ A \} \\ \{ f(t) \mid t \in X \} & \text{otherwise} \end{cases}
\]

Thus, if \( H \) is \( \lambda \)-free, we simply have

\[ f(X) = \{ f(t) \mid t \in X \}. \]

Furthermore,

\[
f(\langle X_1, \ldots, X_n \rangle) = \langle f(X_1), \ldots, f(X_n) \rangle,
\]

\[
f(\Lambda) = \Lambda.
\]

Given a labeled net \( H = (S, A, f) \), we define its 'parallel language' \( pL(H) \) to be

\[ pL(H) = \{ f(w) \mid w \in pL(S) \}. \]

A labeled net \( H = (S, A, f) \) is a 'specification net' iff its labeling alphabet \( A \) is partitioned into an 'input alphabet' \( inH \) and an 'output alphabet' \( outH \), i.e., \( inH \cup outH = A \), and the two alphabets \( inH \) and \( outH \) are disjoint. With a specification net \( H \) we associate the directed process \( P(H) = (inH, outH, pL(H)) \). Given a circuit \( \text{CCT} \) and a specification net \( H \), we write 'CCT imp H' instead of 'CCT sat \( P(H) \)'. 'CCT di-imp H' is defined similarly.
6.3 Algebraic Notations for Petri Nets

In this subsection we introduce a (tentative) algebraic notation for labeled nets, which will be used in this paper instead of the usual graphical representation (see Figure 6.1 for a comparison of the two representations).

Let $S = (P,T,V,M)$ be a marked net. If $V(x,y) = 1$, we say that $x$ is an "input" of $y$, and $y$ is an "output" of $x$. If $M(p) = k$, we say that $p$ "has $k$ tokens". Our algebraic notation for (labeled) nets is as follows.

(a) $[a]$ will denote a net (cf. Fig.6.1(a)) consisting of a single transition labeled 'a', together with a single input place and a single output place. The input place has one token, the output place has none.

(b) An 'entry' place of a net is a place without input transitions and with at least one output transition. An 'exit' place is defined similarly. A '1-1' net has exactly one entry place, exactly one exit place, and no isolated places. The net has one token, namely in its entry place. Let $H_1$ and $H_2$ be 1-1 nets. The 1-1 net $H_1||H_2$ (cf. Fig.6.1(b)) is obtained by identifying the entry places (leaving one token only), as well as the exit places. The 1-1 net $H_1;H_2$ (cf. Fig.6.1(c)) is obtained by identifying the exit place of $H_1$ with the entry place of $H_2$ (after removing the token of $H_2$). We write $[a;b]$ for $[a];[b]$, etc.

(c) Let $H$ be a 1-1 net. The marked net $^*_H$ (cf. Fig.6.1(d)) is obtained by identifying the exit place of $H$ with its entry place (leaving the token there).

(d) Let $H_1$ and $H_2$ be labeled nets and assume that any label of $H_i$ (i=1,2) which differs from $\lambda$, appears only once in $H_i$. Then the labeled net $H_1||H_2$ is obtained from the nets $H_1$ and $H_2$ by identifying any two transitions which have the same label, provided this shared label differs from $\lambda$ (cf. Fig.6.1(e)). The labeled net $H_1#H_2$ is derived from $H_1||H_2$ by replacing by $\lambda$ the label of any transition which has been obtained by identifying a transition of $H_1$ with an (equally labeled) transition of $H_2$.

(e) Let $H_1$ and $H_2$ be 1-1 nets. $H_1&H_2$ denotes the 1-1 net derived from $H_1||H_2$ by the
addition of $\lambda$-labeled transitions, such that $p_L(H_1\&H_2) = p_L(H_1\|H_2)$, as illustrated in Fig. 6.1(f).

7. SOME PROPERTIES OF NETS AND CIRCUITS

In this section we discuss various properties of nets and circuits. These properties are useful for the verification and synthesis of asynchronous circuits, particularly delay-insensitive circuits.

7.1 Some Net Firing Rules

(a) $*[a;z] [z > *[z;a]

(b) $(*[a;z] \| *[b;z]) [(a,b)>$

(*[z;a] \| *[z;b]) [z>

(*[a;z] \| *[b;z])

The above firing rules follow immediately from the corresponding definitions of Section 6.

7.2 Example of DI-Implementation

Proposition 7.1

Let $H$ be a specification net, $H = *[a;z] \| *[b;z]$, where $\text{in}H = \{a,b\}$ and $\text{out}H = \{z\}$. Then

(a) $(CE[A,B;Z],000) \text{ di-imp } H$

(b) $(CE[A,B;Z],11) \text{ di-imp } H$

The proof of this proposition is rather straightforward.

7.3 Some Properties of 'imp'

Proposition 7.2

Let $CCT_1$ and $CCT_2$ be circuits, such that the combined circuit $CCT_1\#CCT_2$ is defined.
Furthermore, assume that \( 'CCT_i \text{ imp } H_i' \) is valid \((i = 1, 2)\), and that \( H_1 \# H_2 \) is defined.

Then

\[ (CCT_1 \# CCT_2) \text{ imp } (H_1 \# H_2). \]

The lengthy proof of this proposition is beyond the scope of the present paper.

We now show that Proposition 7.2 does not hold if the operator \( \# \) is replaced by \( \| \).

Namely, let

\[
\begin{align*}
CCT_1 &= (CE[A,B;Y],000) \\
CCT_2 &= (CE[Y,C;Z],000) \\
H_1 &= *[a;y] \# *[b;y], \text{ in}H_1 = \{a,b\}, \text{ out}H_1 = \{y\} \\
H_2 &= *[y;z] \# *[c;z], \text{ in}H_2 = \{y,c\}, \text{ out}H_2 = \{z\}.
\end{align*}
\]

Then \( CCT_1 \text{ imp } H_i \) \((i = 1, 2)\), but \( (CCT_1 \| CCT_2) \text{ imp } (H_1 \| H_2) \) does not hold, as will be shown below.

We represent a state of \( CCT_1 \| CCT_2 \) by the bit vector \(<A,B,C,Y,Z>\). The following is a state/signal sequence of \( CCT_1 \| CCT_2 \):

\[
00000, \{a,b\}, 11000, \{y\}, 11010, \{a,b\}, 00000
\]

It follows that \(<\{a,b\},y,\{a,b\},y>\) is in \( msb(CCT_1 \| CCT_2) \). Now

\[
(H_1 \| H_2) (\{a,b\})
\]

\[
*[y;a] \# *[y;b] \# *[y;z] \# *[c;z] [y]
\]

\[
*[a;y] \# *[b;y] \# *[z;y] \# *[c;z] [\{a,b\}]
\]

\[
*[y;a] \# *[y;b] \# *[z;y] \# *[c;z].
\]

Thus \(<\{a,b\},y,\{a,b\}>\) is in \( pl(H_1 \| H_2) \), but \(<\{a,b\},y,\{a,b\},y>\) is not. Consequently, \( 'CCT_1\|CCT_2' \) sat \( P(H_1\|H_2) \)' is not valid (in view of Requirement (3) in Subsection 5.2), i.e. \( 'CCT_1\|CCT_2' \text{ imp } (H_1\|H_2)' \) does not hold.

8. DI-IMPLEMENTATIONS OF MARKED GRAPHS

A 'marked graph' is a net in which every place has exactly one input transition and
exactly one output transition. In this section we discuss the synthesis of delay-insensitive
circuits from marked graph specifications. This section is a modified version of material
contained in [YR89]. In order to keep the length of this paper within reasonable limits,
we omit the formal proofs of the results.

8.1 Preliminary Results

Proposition 8.1
Consider the specification net

\[ HCE_n = \{ [a_i; z_n] \mid 0 \leq i \leq n \} , n > 0 , \]

where \( \text{in}HCE_n = \{a_0, ..., a_n\} \) and \( \text{out}HCE_n = \{z_n\} \).

Let \( NCE_n, n > 0, \) be a cascade of \( n \) two-input C-elements, defined inductively as follows.

\[ NCE_1 = CE[A_0, A_1; Z_1] \]

\[ NCE_n = NCE_{n-1} \# CE[Z_{n-1}, A_n; Z_n] . \]

Let \( q_0 \) be one of the two states of \( NCE_n, \) in which all (input, output, internal) variables
have the same value (i.e. all 0 or all 1). Then

\[ (NCE_n, q_0) \implies HCE_n \]

Furthermore, \( (NCE_n, q_0) \text{-dimp} \ HCE_n \).

Proposition 8.2
Consider the specification net

\[ HICE(n, j) = \{ [a_i; z_n] \mid 0 \leq i \leq j \} \}

\[ \{ [z_n; b_i] \mid j < i \leq n \} , n > j > 0 , \]

where \( \text{in}HICE(n, j) = \{a_0, ..., a_j\} \cup \{b_{j+1}, ..., b_n\} \)
and \( \text{out}HICE(n, j) = \{z_n\} \).

Let \( NICE(n, j) = NCE_n \# \{ [\text{INV}[B_i; A_i] \mid j < i \leq n]\} , \)
and let \( q_0 \) be one of the two states of \( NICE(n, j), \) in which all the inputs as well as the out-
put have the same value (0 or 1) , and all the inverters are stable.
Then \((NICE(n,j),q,0)\) \(\text{imp} \ HICE(n,j)\) , as well as
\((NICE(n,j),q,0)\) \(\text{di-imp} \ HICE(n,j)\).

**8.2 Main Result**

The main result of this section is the following.

**Theorem 8.1**

Consider the specification net
\[ H = \{ [x.i:y.i] \mid 1 \leq i \leq n \} \]
which satisfies the following conditions:
1. \([x.1,...,x.n] \cup (y.1,...,y.n) = \text{in}H \cup \text{out}H\)
2. for every \(i, 1 \leq i \leq n\),
   \[ x.i \neq y.i \text{ and} \]
   \[ (x.i \in \text{out}H) \lor (y.i \in \text{out}H) \]
3. any element of \(\text{out}H\) appears at least twice in the above representation of \(H\).

Let \(\text{CCT}\) be the circuit produced by the following algorithm. Then
\[ \text{CCT} \text{ di-imp } H. \]

**Algorithm 8.1**

Let \(\text{out}H = \{z.1,...,z.m\}\). We denote by \(H.j, 1 \leq j \leq m\), the subnet of \(H\) obtained from the above representation of \(H\) by selecting only terms which contain the symbol \(z.j\), i.e.
\[ H.j = \{ [x.i:y.i] \mid (x.i = z.j) \lor (y.i = z.j) \} . \]
We view \(H.j\) as a specification net with \(\text{out}H = \{z.j\}\). Let \(\text{CCT}.j\) be the circuit implementing the specification net \(H.j\) in accordance with Propositions 8.1 and 8.2, provided all the initial input and output values of the \(\text{CCT}.j\) circuits are chosen compatibly, i.e. all 0 or all 1. Then
\[ \text{CCT} = \{ \text{CCT}.j \mid 1 \leq j \leq m \} . \]

**8.3 A Design Example**
In this subsection we illustrate the application of Theorem 8.1 to an example of practical interest. The example is taken over from [YR89].

Example 8.1

In this example we consider the design of the control part of a queue (FIFO) element (cf. [Se80], [Ma86]). Such an element forms part of a pipeline process and its main purpose is to transfer data received from its left-hand neighbor to its right-hand neighbor. We assume that the control module communicates with its neighbors using "four-cycle signalling" [Se80], [Ma86].

The control module in question has inputs A, B from its left-hand and right-hand neighbors, respectively. Its outputs are Y, Z directed towards its left-hand and right-hand neighbors, respectively. The four-cycle signalling assumption requires that the marked graphs *[a;y] and *[z;b] form part of the module specification. Furthermore we assume that the sequencing of the outputs is determined by *[y;z]. In summary, we arrive at the following specification for the queue element control module:

\[ HQC = *[a;y] \parallel *[z;b] \parallel *[y;z] \]

The specification graph HQC satisfies the conditions of Theorem 8.1 and Algorithm 8.1 is thus applicable. Let \( y = z.1 \) and \( z = z.2 \). Then

\[ HQC.1 = *[x;y] \parallel *[y;z] \text{ and } HQC.2 = *[x;b] \parallel *[y;z]. \]

Implementations of HQC.1 and HQC.2 are obtained by means of Proposition 8.2. The final network NQC thus becomes:

\[ NQC = (CE[A,X.1;Y]) \parallel CE[Y,X.2;Z] \parallel INV[Z;X.1] \parallel INV[B;X.2]) \setminus \{X.1,X.2\} \]

The design of a similar circuit is also discussed in [Ma86], [Ch87], [Eb87], and [MBM89]. However, our method is shorter, requires no heuristics, and is proven to be "correct-by-construction". On the other hand, it is applicable only to a suitably restricted family of marked graphs.
8.4 Some Further Results

A few further synthesis results, similar to the above, can be found in [YR89]. They deal with specification nets, formed by additional classes of live and safe (cf. [Pe81]) labeled marked graphs, in which no label appears more than once.

9. FURTHER SYNTHESIS EXAMPLES

In the previous section we derived methods for synthesizing delay-insensitive circuits directly from their marked-graph specification, without the need of transforming the specification into a finite-state machine.

Research is presently in progress with the purpose of extending our direct synthesis approach to larger classes of specification nets. In this section we discuss a few design examples which are intended to serve as starting point for a suitable extension of our design approach.

Example 9.1

Let $H$ be the following specification net

\[ H = ([a \& b];[z]) \uplus ([c];z) \]

where $\text{in}H = \{a, b, c\}$ and $\text{out}H = \{z\}$.

Applying the $\lambda$-elimination techniques discussed in [Yo87b], one easily verifies that the net $H$ may be replaced by

\[ H' = ([a];z) \uplus ([b];z) \uplus ([c];z), \]

yielding the following di-implementation CCT:

\[ \text{CCT} = (\text{CE}(A, B; X) \# \text{CE}(X, C; Z), \ 00000). \]

Example 9.2

Consider now the specification net

\[ H = ([a \& b];[z]) \uplus ([c];z), \]

where $\text{in}H = \{a, b, c\}$ and $\text{out}H = \{z\}$. 
This net is evidently obtained from the net

\[ H' = *(a);(z) \parallel *c, z \]

by replacing \([a]\) by \([a]b\). Such a replacement in \(H'\) corresponds to the substitution of the A-input in the implementation of \(H'\) by a XOR-gate with inputs A and B. We thus get the following implementation of \(H\):

\[ \text{CCT} = (\text{NWK}, \text{q}_0), \text{ where} \]

\[ \text{NWK} = \text{XOR}[A, B, X] \parallel \text{CE}[X, C; Z], \]

and \(\text{q}_0\) is the all-0 or the all-1 initial state of NWK. One easily verifies that 'CCT di-imp \(H'\).

The preceding two examples are easily generalized to any specification net \(H\) of the following type. Let \(H_1\) be a specification net which has a known di-implementation and let \('a'\) be an input signal of \(H_1\). Furthermore, let \(H_2\) be a 1-1 net, derived from input signals which are not in \(\text{inH}_1\{a\}\), by applying the operators \(\cdot\), \(I\), and \(#\). Then \(H\) is the specification net obtained from \(H_1\) by replacing \([a]\) by \(H_2\).

Example 9.3

Let \(H\) be the following specification net:

\[ H = *(\{a \& b\};[y_1] \parallel \{c \& d\};[y_2]);[z]), \]

where \(\text{inH} = \{a, b, c, d\}\) and \(\text{outH} = \{y_1, y_2, z\}\).

This net is a "free-choice net" (cf. [Pe81]). Applying Hack's decomposition algorithm [Ha72], [Ch87], we get the following marked-graph components:

\[ G_1 = *(a \& b);[y_1];[z]) \]

\[ G_2 = *(c \& d);[y_2];[z]) \]

This decomposition yields the following di-implementation:

\[ \text{CCT} = (\text{NWK}, \text{q}_0) \]

where \(\text{NWK} = \text{CE}[A, B, Y_1] \parallel \text{CE}[C, D; Y_2] \parallel \text{XOR}[Y_1, Y_2; Z], \)

and \(\text{q}_0\) is the all-0 or all-1 initial state of NWK.
Example 9.3 can be extended to suitable free-choice nets, the marked-graph components of which have known di-implementations.

10. CONCLUSIONS

We contributed towards a methodology for the direct synthesis of modular, delay-insensitive circuits from suitable classes of specification nets, without the need of transforming the nets into finite-state machines (such a transformation is a major step in the design approaches of [Ch87] and [MBM89]).

Our ongoing research is aimed at extending the applicability of our approach to large classes of specification nets. We also consider the addition of further modules to the basic set of modules discussed in this paper.

The algebraic net-based approach of this paper can be replaced by a suitable process-based approach (cf. [KY88]). Further research on this topic is also in progress.

ACKNOWLEDGEMENT

The author is grateful to John Brzozowski and Jo Ebergen for their very helpful comments.
REFERENCES


Figure 4.1 Example of Network Diagram (NWK)
Figure 6.1 Algebraic Net Notations