Monotonic Circuits with Complete Acknowledgement

Nikolai Starodoubtsev
Tokyo University of Social Welfare
Isesaki City, Gunma 372 0831 Japan
nistarod@ed.tokyo-fukushi.ac.jp

Sergei Bystrov
Institute for Analytical Instrumentation of Russian
Academy of Science St.Petersburg, 198103 Russia
bystrov@iai.rssi.ru

Alex Yakovlev
School of Electrical, Electronic and Computer Engineering
University of Newcastle upon Tyne NE1 7RU, U.K.
Alex.Yakovlev@newcastle.ac.uk

Abstract

The paper studies a class of asynchronous circuits in which every signal transition on the inputs of every gate is acknowledged during the circuit operation. This property is called complete acknowledgement (CA) and it is considered here for circuits that consist of gates described by monotonic boolean functions only. In order to implement such circuits the standard CMOS designs of 2-input logic gates are modified by using an additional output for CA. The paper investigates the behavioral properties of monotonic CA (MCA) circuits and the feasibility of a behavioral specification to be refined to a CA-implementable form. The result of comparison of a number of CA realizations with their speed-independent counterparts produced by negative gate synthesis inspires optimism about the practicality of CA circuits. Being particularly robust to variations in technological parameters, e.g. the value and type of delay and switching thresholds, such circuits offer potential advantages for future CMOS designs.

1. Introduction

Signal transition acknowledgement plays a constitutional role in logic circuit design. Actually there are only two options to make one event, e_1, precede another, e_2 in a circuit: either to arrange its work in such a way that e_1 *causes* e_2 (we also say that e_2 *acknowledges* e_1) or to use physical properties of gates and wires, viz. *delays*, in order to prevent the occurrence of e_2 before e_1. The former underlies the main principle of asynchronous design, the latter is the basis for synchronous one.

As CMOS transistors scale down they become subject to ever-larger statistical variabilities. This causes delay sensitivity to increase quadratically to process variations [1]. Asynchronous design based on acknowledgement being tolerant to process variability is a natural way to meet some short and especially long term challenges in circuit design. Delay insensitivity, absence of the global clock, aptness for reuse and process variability mitigation are inherent for asynchronous circuits. In such a context, interest in design approaches based on more pessimistic assumptions about delays in the circuits increases.

Speed-independent (SI) [2] circuits avoid use of the fundamental mode assumption [3]. They operate correctly under arbitrary delays at the gates' outputs but wires must obey the isochronic fork rules [4]. Delay-insensitive (DI) circuits have most pessimistic assumptions about wire delays between elements, which makes them highly attractive in meeting the technology demands. Despite the fact that the class of DI circuits built of relatively simple logic gates was shown to be limited [5], interest in such circuits has never disappeared. It was particularly warmed up after the proof that indiscriminate use of isochronic forks in SI circuits might not always be safe [4]. Many solutions have been proposed for DI circuits, most of them based on using special DI building blocks, e.g. those from [6]. Often however, these blocks had so complex internal structure that assuming their internal delays to be 'negligible' would be overly optimistic. Finally, there have also been at least two attempts to remove wire delay assumptions at the logic gate level by looking inside the gate structure [7, 8]. In some sense, our present approach is quite close to these.

Asynchronous design itself is asynchronous to a limited extent only: Do we rely on the *fundamental mode* assumption? If not, do we in SI circuits opt to use gates with *arbitrary*, fan-in and fan-out? Do we rely on an *input inverter delay* to be smaller than a path delay in SI circuits? If not, do we rely on the *isochronic fork* assumption? Do we rely on the fact that large enough blocks like C-element, Toggle, XOR gate or other *building blocks* can perform *without haz-*

Proceedings of the Ninth International Symposium on Asynchronous Circuits and Systems (ASYNC'03)
1522-8681/03 $17.00 © 2003 IEEE

ards in delay-insensitive (DI) circuits? Do we finally rely on the assumption that wires can be modelled by *inertial delays*? We start this work with a NO answer to all these questions. Then, another question arises: Is it possible to compose non-trivial circuits of CMOS gates in such a way that every signal transition on the input of every gate is acknowledged by the gate's output, as it happens for example in a simple ring of inverters?

We will show here that for a typical behavior of NAND2 and NOR2 gates three out of four signal transitions on an input are acknowledged on the output. Is it possible to acknowledge all four input transitions? Paper [7] proposed a transistor-level solution to the problem of how to acknowledge on and off switchings of every transistor in such a gate, thereby offering a positive answer to our question. The shortcoming of that solution however was the need to use a pass transistor with lower threshold. Unfortunately, work [7] resulted in a rather pessimistic conclusion: only a narrow class of circuits could be implemented using cells built of RS-flip-flops, constructed out of those special gates. There was no indication that even a C-element's behavior had such a realization.

Another solution to the gate input acknowledgement problem was proposed in [8]. It was based on embedding an inverter into the gate. This allowed every input transition (not every transistor on and off switching!) to be acknowledged. It was crucial that only a slight modification to the standard structure of NAND2 and NOR2 gates was required. Another important feature of this solution was that it used the technique of behavioral refinement, which could put the solution into the overall framework of the behavioral synthesis of circuits with CA. Here, we develop this technique further so as to make it capable of solving tasks of practical importance. We will call such circuits *monotonic circuits with complete acknowledgement* or *MCA circuits* in short.

Dealing only with 2-input gates realizing monotonic boolean functions is a natural way to pursue because such a gate typically has only one input transition unacknowledged. Thus it should be easier to modify to CA form than a more complex, say 3-input gate.

This work summarizes a two-year-long investigation of CA circuits based on refinement techniques initially developed for synthesis of monotonic circuits in [9, 10, 11, 12]. Although a lot of questions still do not have clear answers, today we can definitely make an optimistic conclusion: the class of MCA circuits is wide enough to be useful in practice, which will be confirmed by a complex example. The cost seems to be acceptable. An approach to synthesizing such circuits by means of behavioral refinement and the properties of their behavior are discussed in this paper.

Section 2 introduces a behavior-driven synthesis paradigm, the concept of behavioral refinement and a taxogram language. Two kinds of acknowledgement, direct and indirect, and types of MCA gate connections are discussed in section 3. The possibility of refining an arbitrary initial behavior to an MCA behavior is analyzed in section 4, where different types of behavior are classified. The paper ends up with experimental results (section 5) for a number of behavioral benchmarks and draws conclusions in section 6.

In this paper we will start with an extremely pessimistic assumption about delays in the final circuit and will be trying to reach a maximally optimistic conclusion about its implementability. For that, we will concentrate on investigating the basic behavioral properties of MCA circuits, in order to lay foundations for further development of algorithms for their analysis and synthesis.

2. Behavior-driven synthesis

2.1. Behavior and its refinement

2.1.1 Transitions and states. When the fundamental mode assumption about circuit and environment interaction [3] is not used, one often specifies an asynchronous circuit behavior by *Signal Transition Graph (STG)*. An STG is a Petri net in which transitions are interpreted as positive (from 0 to 1) and negative (from 1 to 0) edges on signals $x_1, x_2, \cdots$ Those are denoted by x_i+ and x_i- (see [13, 14]). STGs gained popularity during the last decade thanks to providing a clear and simple means of presenting *concurrency* and *choice* – two cornerstones of asynchronous circuit behavior. In order to make it possible to observe the dynamics of their behavior, Petri nets and STGs are equipped with the notion of markings (positions of tokens on the arcs or in the places of the net) together with the rules of their evolution (called token game). Marking eventually corresponds to a state of the circuit being specified by an STG.

Our approach to capturing the ordering semantics of STGs will not be based on states. Instead of converting STGs into state graphs, we will use another event-based model, originated from [11], where it was used for circuit synthesis. This technique extracts a set of simpler behaviors from an initial, more complex STG, preserving the original signal transition ordering of the STG. The idea of reducing a behavior to its simple constituents suggested in [11] was successfully applied to some of the tasks in the circuit synthesis flow aimed at a specific gate library. The examples of those were a monotonic gate library [11] and a library with restricted gate fan-in [12], where state-based approaches did not deliver. In this paper we perform the next step and apply this approach to the extreme case of 2-input gates, which must satisfy the additional requirement that input transitions be completely acknowledged for every gate.

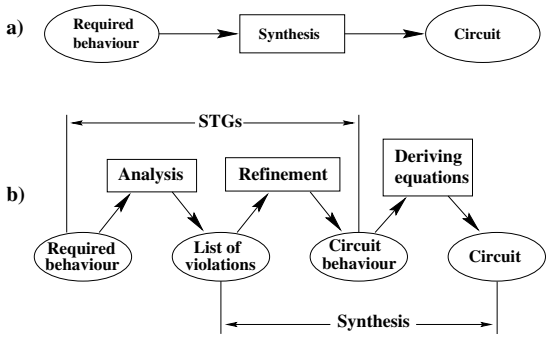

Figure 1. (a) The idea of behavior-driven synthesis and (b) role of behavior refinement

2.1.2 Behavior refinement. The underlying idea of the behavior-driven synthesis procedure is *behavior refinement*, which plays a key role in producing good circuits. The first step in this procedure is to analyze if the behavior to be implemented satisfies certain implementability conditions, e.g. Complete State Coding (CSC) [14, 15]. If it does not, then the second step inserts auxiliary signal transitions in order to make the behavior implementable (see Figure 1). After the refined behavior is fixed one can derive Boolean equations from it. In case of implementability in 2-input gates the emphasis is shifted from deriving equations to the refinement step. Indeed, deriving logic functions of two variables is a trivial task, and as one will see below, easily solvable even without state encoding. Thus, refinement becomes the key stage in getting a circuit with desired characteristics.

2.1.3 Refinable STGs. Keeping in mind that T-nets suggested in [10, 9] are not used today, while STGs, on the contrary, are very popular, we will combine here the approach from [11, 10] with the STG model from [13, 14].

We define a *refinable STG (RSTG)*, by enhancing the traditional STG model with an extra component that distinguishes two kinds of arcs. An arc is *refinable* (represented by a dashed arrow) if it may be replaced by a more complex STG fragment, e.g. a simple event or a chain of events. A *non-refinable or fixed* arc (solid arrow) forbids such a replacement.

If an RSTG contains a *fixed arc* $(r\pm, t\pm)$, where $r\pm$ denotes either $r+$ or $r-$, it means that signal transition $r\pm$ *causes* signal transition $t\pm$ or in the other words $r\pm$ immediately precedes $t\pm$. For the circuit realizing this RSTG, it means that there is a wire connecting the output of gate r with one of the inputs of gate t. A refinable arc $(r\pm, t\pm)$ just means that $r\pm$ precedes $t\pm$ without specifying whether the precedence is immediate or not. So RSTG is regular STG with some arcs declared to be fixed.

On one hand, fixed arcs is an explicit way to say in which arcs signals may not be inserted during synthesis. E.g., for speed-independent circuits they are usually arcs leading from output signal transitions to input ones. On the other hand, the same idea of fixed arcs can be used for emphasizing essential causal relations between signal transitions whereas refinable arcs can be used for hiding irrelevant details of the behavior under analysis or discussion.

2.2. Synthesis and analysis

We will illustrate the idea of synthesis via behavioral refinement using a simple example. Let us consider a Q-element, which decouples two full handshake cycles between with its left and right neighbors. The block diagram of a Q-element is depicted in Figure 2(a), and its behavior is defined by an RSTG in Figure 2(b). Signals a and b are inputs, x and y are outputs.

2.2.1 Q-element analysis. As stated in section 2.1.3 for speed-independent circuits, all arcs leading from output to input transitions must be non-refinable, while all other arcs can be declared refinable. In our transformations of the original RSTG, in order to preserve the required behavior we can only insert transitions of auxiliary signals, which after synthesis will become internal signals in the final circuit. In order to synthesize a circuit, what in our case means

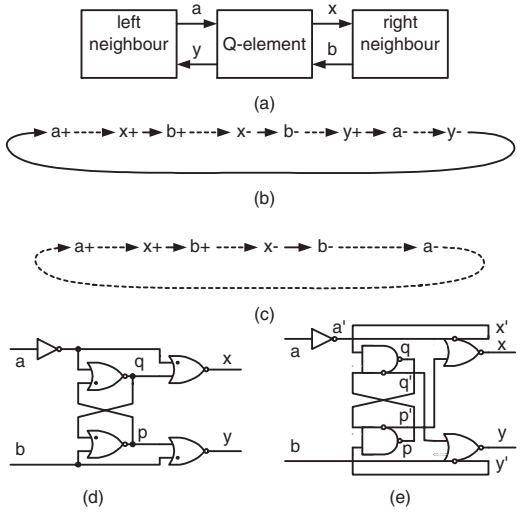

Figure 2. Q_element: (a) block diagram, (b) behavior to be implemented, (c) the first step of behavior analysis, (d) synthesized circuit, (e) CA circuit

to make every signal implementable by a 2-input gate, we will refine the required behavior by inserting auxiliary signal transitions. We start with the **analysis** of the RSTG.

To check whether a signal x can be implemented we delete all signals from the RSTG of Figure 2(b) except for those whose transitions are immediate predecessors to either of the transitions $x+$ or $x-$. In the favorable case x depends on no more than two signals (the gates have two inputs). The RSTG shown in Figure 2(c) is thus obtained. One can see that $a = 1$ and $b = 0$ hold immediately after transition $a+$ as well as immediately after $b-$ whereas signal x should behave differently in these two cases (it should switch to 1 in the first case but be stable in the second). This means that x cannot depend only on a and b. So it is necessary to insert an auxiliary signal transition before $x+$ or $x-$.

2.2.2 Taxograms. Throughout the paper we will analyze RSTGs in a similar way and refine them to the implementable form. For clarity, we will often use a special kind of regular expressions extracted from the RSTGs, which we call *taxograms* (from Greek $\tau\alpha\xi\iota\varsigma$ - order and $\gamma\rho\alpha\mu\mu\alpha$ - letter) following [9]. Here are two examples of taxograms, one corresponding to the Q-element's behavior of Figure 2(b):

$$(a+\cdot\cdot x+b+\cdot\cdot x-b-\cdot\cdot y+a-\cdot\cdot y-)^{\infty}, \quad (1)$$

and the other corresponding to our analysis of signal x shown in Figure 2(c):

$$(a+\cdot\cdot x+b+\cdot\cdot x-b-\cdot\cdot a-\cdot\cdot)^{\infty}. \quad (2)$$

In taxograms dots "$\cdot\cdot$" stand for a refinable arc; no symbols between transitions corresponds to a fixed arc; symbol ∞ means that the sequence of events is infinite so that $a+$ follows $y-$ in (1). This notation of taxograms is a slight modification to that of [9]. Note that taxograms are not intended to replace the (R)STG language. They are used here to simplify and shorten the explanations of properties and transformations essential for MCA circuits.

2.2.3 Analysis and synthesis process. We will only allow *consistent* behaviors to be implemented, for which every signal z must have its transitions $z+$ and $z-$ ordered in a mutually alternating way. Returning to the Q-element example, we need to insert an auxiliary signal transition, say $q+$ before either $x+$ or $x-$ in (1). Let us try, for example, to insert $q+$ before $x-$ and for consistency insert $q-$ somewhere after $a-$, otherwise we would run into the same trouble for signal q as for signal b in (2): $(a+\cdot\cdot x+b+\cdot\cdot q+x-b-\cdot\cdot y+a-\cdot\cdot q-\cdot\cdot y-)^{\infty}$.

Definition. We say that signal x is *2-dependent* if it can be realized by a monotonic 2-input gate, such as NAND2, NOR2, AND2, etc.

When all the signals, except for x and those that cause $x\pm$, have been deleted and arc $(a+, x+)$ fixed, a typical NOR gate behavior is obtained (up to inversion of a):

$$(a+x+\cdot\cdot q+x-\cdot\cdot a-\cdot\cdot q-\cdot\cdot)^{\infty}. \quad (3)$$

Now signal x is 2-dependent. To fix the realization for signal $x = \overline{q+\bar{a}}$ one should fix arcs $(a+, x+)$ and $(q+, x-)$ forever:

$$(a+x+b+\cdot\cdot q+x-b-\cdot\cdot y+a-\cdot\cdot q-\cdot\cdot y-)^{\infty}. \quad (4)$$

In general, where it is not clear whether a particular solution should be fixed or not, it would be better to postpone fixing arcs until all output and inserted signals become 2-dependent. Let us now continue our analysis of (4) for signal y: $(\cdot\cdot b+\cdot\cdot q+\cdot\cdot b-\cdot\cdot y+\cdot\cdot q-\cdot\cdot y-)^{\infty}$. Signal y is also 2-dependent. For analysis of signal q we have: $(a+\cdot\cdot b+\cdot\cdot q+\cdot\cdot b-\cdot\cdot a-\cdot\cdot q-\cdot\cdot)^{\infty}$. One can see that q is not 2-dependent because signals a and b on which q must depend have the values of $a = 1, b = 0$ immediately after $a+$ and similarly after $b-$ while q must have different values in those states. Thus, we need to insert signal transitions in (4) between $b+$ and $b-$ as well as between $b-$ and $b+$. Let us insert an auxiliary p into (4) as follows:

$$(a+x+b+\cdot\cdot p-\cdot\cdot q+x-b-\cdot\cdot \\ y+a-\cdot\cdot q-\cdot\cdot p+\cdot\cdot y-)^{\infty}. \quad (5)$$

One can observe that all non-input signals in (5) are 2-dependent. Let us now fix all previously refinable arcs in (5):

$$(a+x+b+p-q+x-b-y+a-q-p+y-)^{\infty} \quad (6)$$

and derive equations from (6) analyzing signals x, y, q, p one at a time:

$$(a+x+\cdot\cdot q+x-\cdot\cdot a-q-\cdot\cdot)^{\infty} \Rightarrow x = \overline{q+\bar{a}}$$
$$(\cdot\cdot b+p-\cdot\cdot b-y+\cdot\cdot p+y-)^{\infty} \Rightarrow y = \overline{p+b}$$
$$(a+\cdot\cdot p-q+\cdot\cdot a-q-p+\cdot\cdot)^{\infty} \Rightarrow q = \overline{\bar{a}+p}$$
$$(\cdot\cdot b+p-q+\cdot\cdot b-\cdot\cdot q-p+)^{\infty} \Rightarrow p = \overline{b+q}$$

The circuit synthesized for a Q-element is shown in Figure 2(d). It is a standard logic circuit in which for every gate its *passive* (explained in the section 3.1.4) input is marked by a black dot in order to simplify its subsequent transformation to the MCA circuit shown in Figure 2(e).

Finally, the task of circuit *synthesis* can be formulated as follows: it is a process of substituting RSTG fragments with fixed arcs for refinable arcs in such a way that every signal required to be implemented could be realized by a gate with necessary properties. An example of a synthesis process was our *refinement* of behavior (1) to (6).

Conversely, starting from a circuit and its given behavior with all arcs fixed, the *analysis* process is concerned with reducing the given behavior to a simpler one with respect to some signals. In analysis, a fixed arc entering a non-input signal transition is used to present a causality relation. A refinable arc means that a fragment consisting of at least one signal transition was replaced by it (cf. the *reduction* of behavior (1) to (2)).

2.3. Consistent transition deletion and insertion

In order to perform analysis and synthesis we need strict rewriting rules for transforming one RSTG or taxogram to another. There are two consistency conditions: one about replacing an RSTG fragment by a refinable arc (the rule of signal deletion or RSTG *reduction*), and the other about replacing a refinable arc by an RSTG fragment (the rule of signal transition insertion or RSTG *refinement*). These rules are:

consistent signal s deletion If a transition $s\pm$ of signal s is to be deleted (i.e. replaced by refinable arc), then all other transitions of s should be deleted at the same time.

consistent signal s insertion When inserting transitions of an auxiliary signal s, transition $s+$ should follow transition $s-$ and vice versa.

For example, all signal deletions and insertions in section 2.2.3 are consistent.

2.4. Generic taxograms

Sometimes it is not important how particular transitions or their sequences are ordered. We will include them in the braces in order to show that any option of their ordering will do, e.g, $\left\{ \begin{smallmatrix} x+ \\ y- \end{smallmatrix} \right\}$ denotes three available options: $x+y-$, $y-x+$ and $\binom{x+}{y-}$, where the latter stands for parallel transitions $x+, y-$.

3. Complete acknowledgement

Informally, the behavior of a circuit is *Speed-Independent (SI)* if it does not change regardless of the relative values of its *gate delays*. A more formal subclass of SI behavior, which is often identified with SI one, is called *Semi-modular (SM)* [2] or, equivalently, *output-persistent* [15]. The latter allows only transitions of input signals to disable each other, and forbids all other options of disabling. A more restricted class of circuit behavior is called *Delay-insensitive (DI)*, whose informal and 'practical' definition requires the circuit to operate correctly regardless of *wire and gate delays*. In terms of enabling conditions, this means that not only outputs of the gates are disallowed to be disabled (as in SM circuits) but also the ends of delay elements corresponding to all wires[1]. In an alternative formal definition, a DI circuit is one in which for every element every input transition is acknowledged by the element's output before another signal transition is allowed on the same input of the element. We refer to this property as to one of *complete acknowledgement (CA)*.

It is quite evident that CA is a sufficient condition for DI operation. However, up to now there has been no proof of it being necessary in the sense of the 'practical' definition of a DI circuit, i.e. a circuit whose behavior is correct regardless of its gate and wire delays. The key point here is concerned with the delay model used for gates and wires. The traditional delay model in the theory of SI and SM circuits has been that of an *inertial delay* for all gates. It was considered to be more realistic than a *pure* or *transport delay*. On the other hand, for wires there has only been an assumption of *stray delay*(cf. [17]), which does not immediately call for the use of a pure delay model. It is not obvious that such a pessimistic attitude to all wires should be adopted when talking about 'practical' DI circuits. Section 3.3.2 presents an example of the circuit behavior in which some of signal transitions are not acknowledged and yet this behavior is insensitive to wire delays if the latter are considered inertial. The notion of CA defined here helps us to distinguish such 'less conservative' DI cases from 'more conservative' ones. Further, we will focus on *monotonic completely acknowledged (MCA) circuits*.

3.1. Direct acknowledgement

3.1.1 Inverter. A trivial example of behavior with direct acknowledgement is an inverter element. Its single input transition is always acknowledged by its output. Its behavior is specified by the following taxogram: $(a+x-\cdots a-x+\cdot\cdot)^i$, where $i = 1, 2, \cdots$.

3.1.2 Monotonic 2-input elements. Let us consider here NOR and NAND elements with or without inverters on one or both of their inputs and/or their outputs. There are altogether eight *2-input monotonic Boolean functions* that can be implemented, either using a NOR or a NAND. There is no logical difference between such realizations, however, some difference may appear in their dynamic behavior if we use *indirect input acknowledgement*, which will be discussed in section 3.2.

3.1.3 Quasi-inverter. It is a standard monotonic gate (in our case a 2-input one) working in such a mode that its every input transition causes an output transition. When the polarity of signals is important, a *quasi-inverter* is an element in which the output switches in the opposite direction to the input. If the direction is the same, we call it a *quasi-repeater*. For a NOR gate its quasi-inverter behavior can be specified by the following taxogram:

$$((a+x-\cdots a-x+\cdot\cdot)^{j_i}(b+x-\cdots b-x+\cdot\cdot)^{k_i})^i, \quad (7)$$

where $i = 1, 2, \cdots; j_i, k_i = 0, 1; j_i + k_i = 1$.

[1] Note that it is only branching wires, or forks that effectively matter here because the delays of non-forking wires can be attributed to the outputs of the gates.

To any behavior as many quasi-inverter components can be added as necessary without losing the MCA property, so we now concentrate on the behaviors without quasi-inverter components.

3.1.4 Eager transition. One can see that transitions $x+$ and $x-$ on the output of a NOR gate have different nature (see (3)). The first one is conservative: it can take place only after **both** inputs q and $\bar{a}$ become 0. The second one is on the contrary eager to occur in response to **any** of input becoming 1. We call the latter an *eager transition*. For gates NOR and AND $x-$ is eager transition while for OR and NAND it is $x+$.

Definitions. A *signal transition* $a\pm$ is *active* with respect to x (wrt x) if it causes an eager transition $x\pm$. The *signal* a itself is also called *active* in such a case. Otherwise the signal is called *passive*, and one of its transitions, which is not acknowledged by output x, is also *passive*. E.g. in (3) $q+$ is active, $a-$ is passive wrt x.

Any signal that triggers a quasi-inverter is acknowledged by the output and hence active. In a more general case, if at least one transition is passive, it is not acknowledged directly. However, it can be acknowledged indirectly by a built-in inverter if we use an MCA element. It will become clear in section 3.2.3 that only passive signal b can be acknowledged indirectly.

3.2. Indirect acknowledgement

A behavior with every input transition acknowledged either directly or indirectly will be called a *viable behavior*.

3.2.1 Viable NOR2 gate behavior in a non-quasi-inverter mode. We will only consider how a single signal transition or pairs thereof $\cdot\cdot x-\cdot\cdot x+\cdot\cdot$ can be caused. Multiple signal transitions are just a composition of such simple cases.

There are only two ways to cause an eager transition $x-$ for NOR2: either $\binom{\cdot\cdot a+}{\cdot\cdot b+}x-\cdot\cdot$ or $\cdot\cdot a+x-\cdot\cdot$, where transitions in the brackets are executed in parallel. Only the latter case is viable because in the former only one of two transitions $a+, b+$ will be acknowledged.

Let us consider now what can take place after or in parallel with eager transition $x-$. There are the following five options: $\cdot\cdot a+\binom{x-\cdot\cdot}{\cdot\cdot a-\cdot\cdot}$, $\cdot\cdot a+\binom{x-\cdot\cdot}{\cdot\cdot b+\cdot\cdot}$, $\cdot\cdot a+x-\cdot\cdot a-\cdot\cdot b+\cdot\cdot$, $\cdot\cdot a+x-\binom{\cdot\cdot a-\cdot\cdot}{\cdot\cdot b+\cdot\cdot}$ and $\cdot\cdot a+x-\cdot\cdot b+\cdot\cdot a-\cdot\cdot$. In the first two cases $a+$ might be unacknowledged, in the third and forth $a-$ might cause $x+$, which would mean the quasi-inverter mode. So only the last sequence is viable when one of the two NOR2-gate inputs is passive.

Finally, note that only one signal transition of active signal a is allowed between $x-$ and $x+$, where the former is an eager transition, otherwise the first signal transition $a-$ is not acknowledged, i.e. behavior $\cdot\cdot a+x-\cdot\cdot a-\cdot\cdot x+\cdot\cdot$ is viable while $\cdot\cdot a+x-\cdot\cdot a-\cdot\cdot a+\cdot\cdot a-\cdot\cdot x+\cdot\cdot$ is not. Summarizing the above one can see that only those behaviors are *viable for a NOR gate* x with one input passive that are captured by the following taxogram:

$$(\cdot\cdot a+x-\cdot\cdot(b+\cdot\cdot b-\cdot\cdot)^n b+\left\{\begin{matrix}\cdot\cdot a-\\ \cdot\cdot b-\end{matrix}\right\}x+)^\infty, \quad (8)$$

where fragment $b+\cdot\cdot b-\cdot\cdot$ can repeat n times, $(n=0,1,2,\cdots)$.

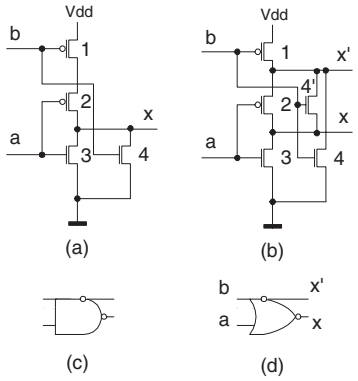

Figure 3. Circuits of usual NOR (a) and caNOR (b) gates, symbols of caNAND (c) and caNOR (d) gates

3.2.2 Behavior equivalence. Two behaviors are *equivalent* if one can be converted to the other by applying the following transformation: (i) one-to-one renaming some signals, (ii) cyclic transposition within external brackets "()", and (iii) swapping signs for **all** signal transitions of any signal.

For example, the following behavior allowed by (8): $(\cdot\cdot a+x-\cdot\cdot b+\cdot\cdot a-\cdot\cdot b-x+)^\infty$ is equivalent to (3) as well as to every behavior derived from (6) in section 2.2.3.

Statement. Viable behaviors that can be implemented by an MCA circuit are those of either an inverter or a quasi-inverter (7) or an MCA gate (8), or a concatenation of those or the behaviors equivalent to them.

A typical case of (8) is when transition $a+$ is acknowledged by $x-$. Both transitions $a-$ and the last $b-$ are acknowledged by $x+$ irrespective of the order of their arrival at the inputs of x. All transitions $b+$ and non-last $b-$ are not acknowledged directly. Thus, our analysis and synthesis of a MCA circuit will be based on checking if every interval $x-\cdot\cdot x+$ for every signal x satisfies one of conditions (7, 8) with subsequent correction by refinement if necessary.

3.2.3 MCA gate. A CMOS construct for a NOR-type MCA gate is shown in Figure 3(b). We call it *caNOR* gate.

This gate differs from the classical CMOS NOR2 gate, depicted in Figure 3 (a), by having an embedded inverter. This inverter shares one transistor with the gate structure (see Figure 3(b)), thus providing output x' in addition to the main output x. This extra output helps indirect acknowledgement of the passive signal transition $b+$. Symbols for caNOR and dual to it *caNAND* are depicted in Figures 3 (d) and (c) respectively. In the rest of the paper we concentrate on the properties of the behaviors implementable by circuits built of such gates with embedded inverters and do not discuss the pros and cons of any particular realization of such gates.

In order to see how these MCA gates work one can consider the MCA implementation of a Q-element shown in Figure 2(e). The latter has the following detailed behavior: $(a+a'-x'+x+b+y'-p+q'-q-p'+x-b-y'+y+a-a'+x'-q+p'-p-q'+y-)^{\infty}$, with the polarity of signals p and q changed to the opposite when performing circuit minimization based on deMorgan's laws [8]. Note that this circuit contains only point-to-point connections being typical for MCA circuits without parallelism.

The results of analogue (HSPICE) simulation of some circuits built of such gates have been presented in [8]. They proved working well and showed remarkable robustness to supply voltage variations.

One can appreciate the role of *indirect acknowledgement* and the exact difference between SI and MCA circuits by comparing the realization of signal x in the two circuits shown in Figure 2. In the SI circuit we **rely** on the passive transition $a-$ being accepted by gate x (more exactly, on its p-transistor turning off) earlier than the $q-$ transition occurs, otherwise a glitch $x+$ on the output x may appear (see fragment $a-q-$ in (6)). In the MCA circuit we are **sure** that transition $a-$ is first accepted by the built-in inverter x' (and, hence, its p-transistor, shared with gate x, is already turned off) and, only after that signal transition $p'-$ can reach the input of gate x (see fragment $a-a'+x'-q+p'-$ from the Q-element MCA behavior). With any p-transistor of the x gate being off a glitch on $x+$ becomes impossible.

3.3. MCA gate connections

First of all the behavior to be implemented by an MCA element should satisfy the viability conditions (8). When the type of wire delay is not defined explicitly both inertial and transport delay models will be applicable.

3.3.1 Asymmetric MCA gate connection. This connection can be only applied to an element whose behavior satisfies condition $n = 0$ for taxogram (8), in particular having only two transitions $b\pm$ per cycle. In addition, at least one transition of signal b should not be passive. The example in Figure 4 shows the corresponding circuit fragment. Thus, in order to satisfy indirect acknowledgement for signal b the latter must be an active signal for some signal c. In this case the behavior of signal x should be e.g. as follows: $a+x-\cdots b+c-\cdots\binom{a-}{b-}x+\cdots c+\cdots$. In order to indirectly acknowledge positive transition $b+$ we refine this behavior to $a+x-\cdots b+x'-c-\cdots\binom{a-}{b-x'+}x+\cdots c+\cdots$, where $b+$ is only acknowledged by an embedded inverter x', not by the original output x of the gate. This type of *acknowledgement* is called *indirect*.

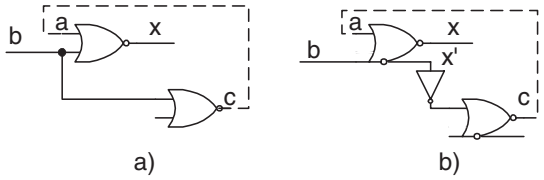

Figure 4. 2-input gate transformation to CA element: (a) fragment of 2-input-gate circuit (b) corresponding CA circuit

Remark 1: If signal b is passive in relation to more than one gate then all such gates are connected as a chain of embedded inverters terminated by a single active input.

Remark 2. Active and passive signals a and b may never exchange their roles. The implementation of an MCA element makes indirect acknowledgement available only for a passive signal b. So if we have two or more different pairs of transitions $x\pm$, then either both signals a and b are active (quasi-inverter mode) or only one signal b can be passive.

Let us consider now two types of symmetric MCA gate connection, dual and quadric.

3.3.2 Quadric CA gate connection. This form is used when a passive signal b is *multiple*, i.e. more than two transitions $b\pm$ occur between $x-$ and $x+$. The basic idea is to insert extra signals in order to 'distribute' the acknowledgment of b between them. A typical behavior pattern is as follows: $a+x-\cdots b+\cdots b-\cdots b+\cdots a-\cdots b-x+\cdots$. We refine it to: $a+x-\cdots b+y-\cdots y+\cdots b-z+\cdots z-\cdots b+w-\cdots w+\cdots a-\cdots b-x+\cdots$, where a number of extra signals, y, w and z, are used to acknowledge individual transitions of b. After b is acknowledged by them it arrives on the input of element x at the time when a is at 0, and hence the last $b-$ transition is acknowledged by x. The corresponding circuit is presented in Figure 5 (b), where signal b is connected to the built-in inverter shared by four MCA gates. In Figure 5 (b) this shared inverter is depicted as a couple of two built-in inverters, whose inputs and outputs are connected in parallel. (For CMOS circuits such a connection is functionally equivalent to a single CMOS inverter). This construct 'insulates' the multiple signal b from the rest of the circuit. The latter sees only non-multiple signals x, y, z, w.

If a multiple signal b has more than four transitions per

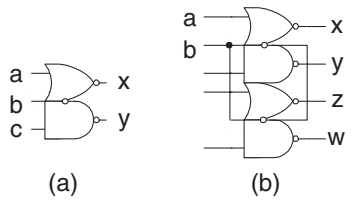

Figure 5. Indirect symmetric acknowledgement: (a) dual, (b) quadric

the cycle, say six: $b+\cdot\cdot b-\cdot\cdot b+\cdot\cdot b-\cdot\cdot b+\cdot\cdot b-\cdot\cdot$, then this multiplication can be reduced to: $b+f-\cdot\cdot f+\cdot\cdot b-g+\cdot\cdot g-\cdot\cdot b+f-\cdot\cdot f+\cdot\cdot b-g+\cdot\cdot g-\cdot\cdot b+h-\cdot\cdot h+\cdot\cdot b-i+\cdot\cdot i-\cdot\cdot$. The additional signals f, g, h, i, some of them being multiple, insulate b from the context. But their multiplicity is less than that of b, plus signal b drives only four gates instead of six. More exactly, it drives only one built-in inverter that is shared by four gates instead of six.

Inertial wire delay. Let us assume that wire delays associated with multiple signals are inertial. Consider a multiple signal a that is active but we do not require all transitions of input a to be acknowledged by signal x while its other input, b, keeps output x stable, e.g.: $a+x-\cdot\cdot b+\cdot\cdot a-\cdot\cdot a+\cdot\cdot a-\cdot\cdot b-x+\cdot\cdot$. Thanks to the inertial delay assumption, this incomplete acknowledging does not however imply hazards on output x. Such a circuit might be called DI (wrt to inertial delays) but it is not an MCA one.

Remark 1. Saying the delay of wire a is inertial effectively refers to the property of the line connecting the gate a output to an input of gate x.

Remark 2. The application of the inertial delay model helps avoid the use of a quadric MCA gate connection, not only for wires inside the circuit but also for primary input signals. The latter are more likely to be carried by a line with transport delay. In such a case it should be sufficient to put an input repeater for this signal before it forks for transforming its transport delay into inertial one.

From now on we will not consider inertial delay. Therefore a quadric MCA gate connection might be needed to make every input transition directly or indirectly acknowledged.

3.3.3 Dual CA gate connection. We have considered above the following two cases: (i) signal s is not multiple and at least one of its transitions is not passive; (ii) signal is multiple and, hence, passive. Now let us consider the case of signal b being not multiple and always passive. A typical behavior for this case is: $\binom{a+x-}{c+}\cdot\cdot b+y-\cdot\cdot\binom{c-y+}{a-}\cdot\cdot b-x+\cdot\cdot$. The circuit depicted in Figure 5(a) guarantees that signal b is acknowledged by output y (resp. x) only after it has been accepted by an input transistor of element x (resp. y).

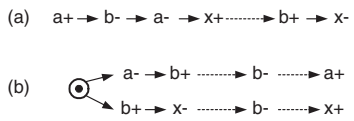

Figure 6. DI Violations due to environment: (a) without choice - DIV1, (b) while choice - DIV2

Statement. Any circuit consisting of 2-input gates satisfying the requirements of (7) and (8), up to equivalence-preserving transformations, can be transformed to an MCA circuit according to the rules given in the above section.

4. Refinement to MCA behavior

In order to implement a behavior by an MCA circuit it should satisfy the acknowledgement conditions for all of its signals. We have so far considered restrictions for internal and output signals transitions. Here we are considering requirements to input signal transitions. We will distinguish three kinds of possible violations: *SI violation* if the SI condition [15] is not satisfied; *DI violation (DIV)* if the SI condition is satisfied but DI condition is not; *MCA violation (MCAV)* if DI condition is satisfied but MCA implementation does not exist. Two typical cases of DI violation are discussed below:

4.1. Behaviors having no realization by DI circuits

We will consider typical cases by illustrating them with simple examples.

4.1.1 Violation without choice (DIV1). This violation takes place if there are no output transitions between two transitions of the same input signal a and at least one transition of a causes an output transition $x\pm$. An example is $(a+b-a-x+\cdot\cdot b+x-\cdot\cdot)^\infty$, where a, b are input and x output signals, as shown Figure 6(a). This kind of violation has some benchmark examples presented in Table 1: `sendr-done, sbuf-send-ctl` (they are also non-SI), `rpdft` and `sbuf-send-pkt2`. There are no means in a DI circuit to make sure that transitions $a+a-$ have occurred. In order to distinguish the state preceding $a+$ from the state following $a-$ it is necessary to insert a signal transition between $a+$ and $a-$, which would violate the original behavior specification.

4.1.2 Violation with choice (DIV2). This violation takes place if: (i) there are two input signals a and b and (ii) the transition of b determines the moment of making choice while signal a, by its level, determines the direction

of choice (signal a gets stable before b makes a transition), and (iii) there are no output signal transitions between a gets stable and b makes the transition which determines the selection point. Figure 6(b) explains this behavioral pattern. The lower branch is taken when a stays at its initial level 1. A DI circuit cannot distinguish the case of the absence of $a-$ from that of $a-$ having occurred but not yet received at the circuit's terminal. (DI circuits inherently have not means of detecting the absence of an event.)

The `rcv-setup` and `sbuf-send-ctl` examples from Table 1 have this kind of violations. Combinations of such violations can take place. It is impossible to build a DI circuit for them without putting certain constraints (cf. non-equivalences used in [16]) on the circuit's and environment's delays. Wherever necessary, minimally restrictive delay constraints have been put in our benchmark realizations. A special form of taxograms is used for specifying behavior with choice. E.g., the behavior shown in Figure 6(b) is defined by $\frac{a-b+\cdot\cdot b-\cdot\cdot a+}{b+x-\cdot\cdot b-\cdot\cdot x+}$.

4.2. Separable DI behavior

We call a behavior *separable* if for every choice option (branch) there is a unique input signal which determines the choice. In practice such signals act as branch or mode identifiers (cf. signals DSr and DSw in the widely used VME-bus controller example of [15]).

4.2.1 Behavior without choice. One subclass of such a behavior that is implementable by CA circuits was found in [7]. The exact limit of behavior implementability by MCA circuits is not established yet, however, one can be sure that it is wide enough to be useful in practice. One rather practical example, `master-read`, is depicted in Figure 7 for illustration. We can conjecture that an SI behavior without choice and which is free from DIV1 can be realized by an MCA circuit.

4.2.2 Behavior with choice. Recall the DIV2 case, i.e. the choice can only be caused by a transition on an input signal, but not by the absence thereof. In our analysis of the STG we should consider pairs of the so called virtual flows.

Definition. *Virtual flow* is obtained from the STG by its traversal through all choice points and consistent random deletion at every choice point of all the flows (branches) except one. At the first choice point, however, two arbitrary branches must be preserved.

This transformation will result in a model with a single (starting) choice point. In further consideration wherever we use term "flow", we will actually mean virtual flow. Note that the virtual-flow-based approach has nothing to do with the actual method used for synthesis of MCA circuits; it is used only for analysis of some properties. This sort of analysis is similar to the idea of linear-time semantics analysis, which assumes that all choices in the model are resolved at the start of the execution.

The underlying refinement idea is to isolate one flow from all others in order to make common signals invisible for every flow after refinement. For a separable DI behavior every signal z that enters several different flows does not determine a choice: $\frac{\cdot\cdot z-\cdot\cdot z+\cdot\cdot}{\cdot\cdot z-\cdot\cdot z+\cdot\cdot}$. We use the following refinement in this case: $\frac{\cdot\cdot f+z-h+\cdot\cdot h-\cdot\cdot f-z+i-\cdot\cdot i+\cdot\cdot}{\cdot\cdot g+z-j+\cdot\cdot j-\cdot\cdot g-z+k-\cdot\cdot k+\cdot\cdot}$, where f, g, h, i, j, k are signals inserted during refinement.

In this case the common signal z is implemented by a quasi-inverter, which collects unique signals f and g for every flow. A quadric MCA gate connection should be used in order to accept signal z on passive inputs. The pairs h, i and j, k of outputs should go back to their own flows, like it was done when we dealt with multiple-transition signals. Recall that this technique works correctly only if signal z does not determine the choice flow.

Statement. With every common signal insulated by signals that are individual for each flow, one can consider each flow as a stand-alone behavior without choice.

Analysis of inseparable DI behavior comes out of the scope of this paper, for example analysis of the case where choice is determined by a signal that is shared by two or more flows. In those cases there are certain conditions when an MCA implementation may be obtained by applying signal refinement. There is, however, a condition, which prevents MCA implementation of inseparable DI behavior. In table 2 it is referred to as an MCA violation or MCAV.

5. Experimental results

Some experimental results are presented in Table 1. They help estimate the cost of using 2-input and CA circuits. Although examples without choice are easier to implement we have focused on those with choice because of their practical importance. They have interesting properties and are also theoretically more challenging. Some of these examples have DIV1, DIV2, MCAV or SI violations and cannot be realized by a pure CA circuit without some delay restrictions. In such cases the tactics of employing the weakest possible restrictions was used.

The leftmost column shows the name of an example. The next three columns show size estimates in terms of the number of CMOS transistors. This is given for monotonic circuits obtained by applying a behavior refinement technique for MCA circuits (the second column), SI monotonic circuits built of two-input gates only (the third column) and built of arbitrary gates (the fourth column). The rightmost two columns show violation types (if any) as well as the number of delay constraints to be satisfied if a violation takes place. E.g., for the circuit `sbuf-send-pkt2` the implied delay constraint was: $D^0_w(rejpkt, f) < D_e(rejpkt-, req+) + D^1_w(req, f)$.

Table 1. Experimental data

Example	Size (number of transistors) via monotonic refinement			Violations	
	MCA circuits	2-input gates	arbitrary	Violation class	Number of implied non-equvalences
sendr-done	30	26	20	non-SI	3
chu-172	30	20	20	no	0
sbuf-send-pkt2	112	46	52	DIV1	1
rpdft	80	48	44	DIV1	3
sbuf-send-ctl	63	50	68	non-SI+DIV2+MCAV	3+1+1
rcv-setup	34	34	34	DIV2	2
vme-read-write	134	74	72	no	0
master-read	208	166	166	no	0
alloc-outbound	58	54	60	no	0
tsend-bm	235	138	138	no	0
C-element	32	30	14	no	0
Q-element	22	18	18	no	0

In order to roughly estimate what new features MCA circuits bring compared to standard SI circuits we present four realizations of the `master-read` example in Table 2. The rightmost two columns show data for the `mr1` circuit synthesized by Petrify [15], which is functionally similar to `master-read`. The average load was calculated without taking into account inverters and external load. For all gate inputs, the load was taken as 1, excepting the load of the MCA gate's passive input, which was 1.1 due to the third ('trickle') transistor $4'$ inside the MCA gate (see Figure 3). The size of the 2-input implementation of C-element was supposed to be 30 transistors, i.e. the smallest known today. The data for the case when C-element is implemented by a non-standard circuit of 8 transistors with weak feedback inverter are presented in the brackets "()".[2] One can see that the MCA realization offers many positive features, relying only on the assumption that a monotonic gate cannot produce a non-monotonic waveform on its output if its input signals rise or fall monotonically, or in other words "there no hazards inside monotonic gates".

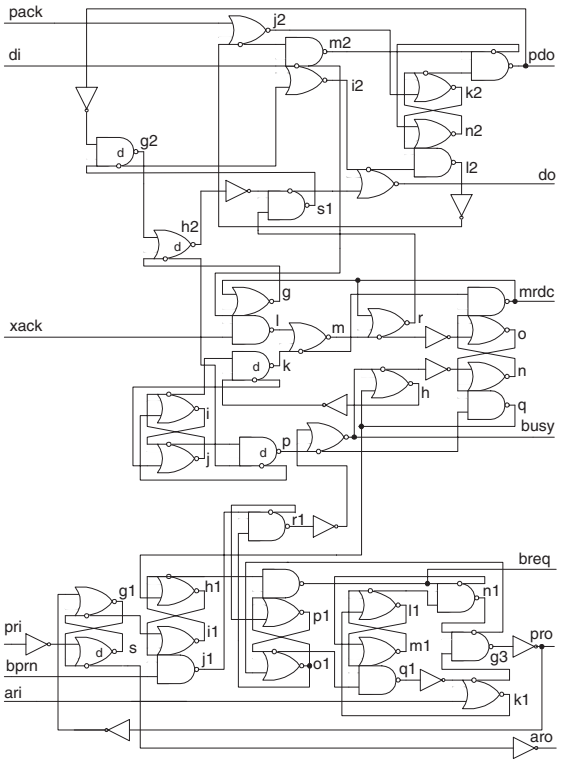

Figure 7. Master-read example

6. Conclusion

This work advances the approach to constructing asynchronous circuits based on behavioral monotonic refinement which originates from [10, 11]. We have tackled here perhaps the most challenging aspect of delay-insensitivity, the case of circuits built of monotonic 2-input gates where every input signal transition is acknowledged directly or indirectly. Standard STGs and T-nets have been unified into the language of refinable STGs, appropriate for defining various refinement rules and operations. The conditions

[2] We, however, avoided in our work consideration of such circuits with weak inverters

Table 2. `Master-read` **realizations**

Feature	Monotonic		Petrify's	
	MCA	2-inp.	2-inp.	Random
Max fan-in	2	2	2	6
Max fan-out/load	2	3	5	7
Avrg fan-out/load	1.07	1.73	1.73	2.75
Size (trans. no.)	208	166	384 (186)	202 (160)
Non-standard gates	yes	no	no (yes)	yes
Postulates presumed:				
No hazards inside monotonic gates	yes	yes	yes	yes
Isochronic forks	no	yes	yes	yes
Inertial delays	no	yes	yes	yes
Zero inverter delay	no	no	yes	yes
No hazards inside building blocks	no	no	no (yes)	yes
Fundamental mode	no	no	no	no

of the implementability of a DI behavior by MCA circuits have been investigated. Some favorable and unfavorable cases are now established. Applying this approach to behaviors with choice raises optimism regarding the MCA-implementability of most practical patterns of behavior, which are not denied by the nature of acknowledgement itself. This work is under development and does not present a 'closed book' on the subject as well as formal proofs of all statements. However, these ideas, examples and solutions lead to a rather optimistic conclusion. Firstly, a monotonic circuit with a typical case of about 75% of input transitions acknowledged directly and 25% indirectly makes MCA designs look promising. Secondly, the behavioral refinement technique is the right way for synthesizing such circuits and for now is probably the only visible on the horizon.

The authors express careful optimism about the future of such circuits. It is *careful* because (i) not all statements are strictly proven yet, and (ii) it is not clear yet if industry will be happy to use the proposed MCA elements. It is *optimism* because (i) there are some good examples of MCA (or almost MCA) realizations for rather complicated behaviors, (ii) there has been no case of practically important behavior found yet that has no MCA realization, and (iii) the applicability of the results of this work not only to MCA circuits but to standard 2-input-CMOS-gate circuits with about 75% of input transitions acknowledged.

References

[1] International Technology Roadmap for Semiconductors (ITRS), 2001 Edition, http://public.itrs.net/Files/2001ITRS/Home.htm

[2] David E. Muller and W. S. Bartky. A theory of asynchronous circuits. In *Proceedings of an International Symposium on the Theory of Switching*, pages 204–243. Harvard University Press, April 1959.

[3] M.B. Josephs, S.M. Nowick and C.H. van Berkel. Modeling and design of asynchronous circuits. Proc. IEEE, vol. 87, no. 2 (Special issue on Asynchronous Circuits and Systems), pp. 234–242, February 1999.

[4] Kees van Berkel. Beware the isochronic fork. Integration, the VLSI journal, 13(2):103-128, June 1992.

[5] A. J. Martin. The limitations to delay-insensitivity in asynchronous circuits. In William J. Dally, editor, Advanced Research in VLSI, pages 263-278. MIT Press, 1990.

[6] R.M. Keller. Towards a theory of universal speed-independent modules, IEEE Trans. Comp. 23(1974), pp. 21-33.

[7] V.I. Varshavsky. Circuits insensitive to delays in transistors and wires. Digital Systems Laboratory, Helsinki University of Technology, Series B: Tech. Rep., No.7, Nov. 1989.

[8] N.Starodoubtsev and A.Yakovlev. Isochronic fork-free asynchronous circuits. In: *9th UK Asynchronous Forum.* Univ. of Cambridge, Cambridge, December 2000, pp. 55-60.

[9] N.A.Starodoubtsev. Synthesis of Control-Dominated Circuits for Parallel Computing Systems. Leningrad, Nauka, 1984, 191 c. (In Russian)

[10] N.A.Starodoubtsev. Asynchronous processes and antitonic control circuits. Part I. Description language. *Soviet Journal of Computer and System Science (USA)*, 1985, vol.23, No.2, pp.112-119.

[11] N.A.Starodoubtsev. Asynchronous processes and antitonic control circuits. Part III. Realization. *Soviet Journal of Computer and System Science (USA),* 1985, vol.24, No.2, pp.44-51.

[12] N.Starodoubtsev, S.Bystrov, M.Goncharov, I.Klotchkov and A.Smirnov. Toward synthesis of monotonic Asynchronous circuits from signal transition graph. In: *Proceedings of the Second IC on Application of Concurrency to System Design ACSD 2001*. Newcastle upon Tyne, June 2001, pp.179-188.

[13] L.Y.Rosenblum, and A.V.Yakovlev. Signal graphs: From self-timed to timed ones, in *Proc. Int. Workshop Timed Petri Nets*, Torino, Italy, 1985, pp. 199–207.

[14] T.A.Chu, C.K.C.Leung and T.S.Wanuga. A design methodology for concurrent VLSI systems. In: Proceedings of ICCD, 1985, pp.407-410.

[15] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A.Yakovlev. Logic Synthesis of Asynchronous Controllers and Interfaces, Springer-Verlag, Berlin, 2002.

[16] N. Sretasereekul and T. Nanya. Eliminating Isochronic-Fork Constraints in Quasi-Delay-Insensitive Circuits ASP-DAC 2001 (Asia and South Pacific Design Automation Conference 2001), Yokohama, Japan, Januray 2001.

[17] K.S. Stevens. Practical verification and synthesis of low latency asynchronous systems, PhD Thesis, University of Calgary, September 1994.