14th IEEE International Symposium on Asynchronous Circuits and Systems

Automated verification of asynchronous circuits using circuit Petri nets

Ivan Poliakov, Andrey Mokhov, Ashur Rafiev, Danil Sokolov, Alex Yakovlev
{ivan.poliakov, andrey.mokhov, ashur.rafiev, danil.sokolov, alex.yakovlev}@ncl.ac.uk

***Abstract*—To detect problematic circuit behaviour, such as potential hazards and deadlocks, in a reasonable amount of time a technique is required which would avoid exhaustive exploration of the state space of the system. Many of the existing methods rely on symbolic traversal of the state space, with the use of binary decision diagrams (BDDs) and associated software packages. This paper presents an alternative approach of using a special type of Petri nets to represent circuits. An algorithm for automatic conversion of a circuit netlist into a behaviourally equivalent Petri net is proposed. Once the circuit Petri net is constructed and composed with the provided environment specification, the presence and reachability of troublesome states is verified by using methods based on finite prefixes of Petri net unfoldings. The shortest trace leading to a deadlock or a hazard in the circuit Petri net is mapped back onto the gate-level representation of the circuit, thus assisting a designer in solving the problem. The method has been automated and compared against previously existing circuit verification tools.**

I. Introduction

The design of an asynchronous circuit usually consists of three distinct stages: *specification*, *implementation* and *validation* of a circuit.

First, the designer determines a specification of a circuit as an unambiguous description of its expected behaviour. A circuit can be specified using various formalisms, of which Signal Transition Graphs (STGs) [5], [24] is currently one of the most popular models [30].

Once the desired specification is constructed, its implementation needs to be obtained. This may be accomplished either manually or by using automated synthesis techniques, or both, which is more realistic. For STGs tools like Petrify can be used [6]. Finally, the obtained circuit implementation needs to be validated against its specification to ensure its correctness before it is passed to later design stages. A designer can usually choose between two methods of circuit validation: *simulation* or *formal verification*.

Simulation can be used to demonstrate correct functionality of a circuit under certain stimuli from the environment. However, this cannot reveal *all* of the possible circuit behaviours since it would require examination of all allowable sequences of actions of the environment, which quickly leads to combinatorial explosion problem.

The formal verification of an asynchronous circuit applies different schemes to prove that the circuit does not exhibit incorrect behaviour following any possible input sequence. Unlike simulation, the verification methods do not explicitly enumerate the input sequences, thus avoiding the combinatorial explosion.

Simulation and verification are particularly different in their results for asynchronous circuits, because the latter often exhibit a high degree of concurrency. Moreover, the environment's choice of input signal transitions can be concurrent with the internal signal transitions, thus making techniques such as cycle accurate analysis ineffective. In those circumstances, the complexity of validation by simulation increases, and demands for the use of analytic exploration of the behavioural models of the circuit implementations. It is therefore imperative to consider formal verification using models similar to those used for specification. This paper pursues this approach by combining both specification and implementation in one formalism, that of Petri nets or STGs.

Note that designs of relatively large circuits are often hierarchical and compositional. Therefore, individual blocks of such designs are built after their sub-blocks have been designed and validated. Appropriate forms of interface of the sub-blocks are required, in which the behavioural complexity of the internal implementation of sub-blocks is hidden behind a subset of interface signals. For example, one can abstract away from the timing conditions used inside the sub-blocks, thereby considering the system at the higher level of abstraction from the point of view of its speed-independent behaviour. Conversely, the design may assume a block to be operating in a speed-independent context but the actual internal behaviour of the implementation needs to be validated in terms of its freedom from hazards.

Compositional approach can be achieved in different ways depending on the modelling method used for verification. For example, in the context of Petri nets, a block whose implementation satisfies its Petri net based specification, can be 'substituted' inside a more complex design not by the Petri net model of its implementation but rather by the Petri net specification. The specification can be significantly more compact, thereby helping to reduce the complexity of analysis at the higher level. This is one of the main motivations for our approach, and this paper tackles some of its aspects.

Similar views were pursued in the development of the verification method underlying the tool Versify [21]. This method checked the correctness of a gate level implementation of a circuit against its STG specification, by considering the closed system whose state space was subject to analysis of undesirable conditions. However, the closed system was formed not by converting the circuit to the same modelling language as the specification as we propose here. It was formed implicitly, at a symbolic state-space traversal stage, where both the gate-level netlist (i.e. set of Boolean equations) and the

Authorized licensed use limited to: Newcastle University. Downloaded on May 26,2010 at 15:37:51 UTC from IEEE Xplore. Restrictions apply.

specification STG contributed to the respective state vector components.

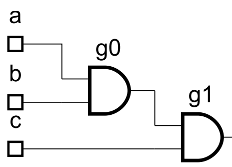
Due to the methodological similarity in both approaches from the point of view of the hierarchical verification flow, it is natural that our method is compared to that of Versify. The benchmarks results show a significant (a least by an order of magnitude) advantage in computation time of our approach compared to that of Versify. This advantage is especially prominent if the circuit exhibits a high degree of concurrency, e.g. circuits consisting of a number of similar cells working in parallel. While the state space grows exponentially, the size of unfoldings for this class of circuits grows linearly.

We should also put this work in context with other approaches published recently on the subject of formal verification of asynchronous circuits [2], [31], [12], [23], [18]. The verification approaches can be categorised by i) the use of timing information (untimed vs timed); ii) representation of the circuit state space (reachability graph vs BDD vs unfolding prefix); iii) circuit types, subdivided according to their functionality (control vs data path) or size (small vs large controllers). The specific nature of our verification approach is that we tackle here untimed control circuits, potentially large scale controllers (up to 100s of signals) and use a Petri net unfolding prefix. This method can be in the future extended to circuits with timing constraints and generalised relative timing (cf. [25], [18]), where certain class timing conditions can be presented as Petri net constructs for concurrency reduction [4].

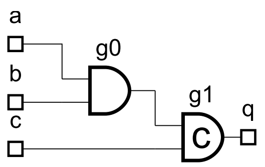
Another potential advantage of our approach based on the unified modelling formalism of Petri nets is its applicability, without much change in the main verification engine, at higher design levels, such as system architecture level. We are using it to model and verify complex behavioural semantics of Static Data Flow Structures (SDFS) [20], [26], [27]. Here notions of early propagation, counterflow and antitokens are easily captured ([1], [3]). For example, we have been able [26] to detect the possibility of hazards in the untimed domain for the implementation of a counterflow pipeline controller [1].

To summarise, the full potential of the presented approach, based on circuit Petri nets, can probably be fully assessed from the analysis of its usefulness in a range of representation levels, from those of SDFS to gate-level netlists, and even their combinations, something which, to the best of our knowledge, hasn't been successfully achieved before. A software environment, called Workcraft, which encompasses this unified modelling and analytic framework, with interfaces to specific simulation, verification and synthesis, tools is now being developed by us.

The paper is organised as follows. Section II contains a motivating example; section III introduces the notion of circuit Petri nets; section 4 presents a method of converting circuits into Petri nets; section V explains how the verification is done; sections VI the overviews the automation of the method in Workcraft framework; section VII compares the proposed method to the currently available tools; sections VIII and IX highlight the ideas for future research and summarise the current contribution.



(a) 2 AND gates



(b) AND and synchronising C-element

Figure 1. An intuitive implementation of 3-input AND gate

II. Motivation

Asynchronous circuit design is considered to be very difficult, and this is one of the major drawbacks of asynchronous circuits when compared to the more traditional synchronous designs. Let us consider an example shown of Figure 1 (a), which is a possible implementation of a 3-input AND gate. Intuitively, one would think that since $((a \wedge b) \wedge c) = (a \wedge b \wedge c)$, this circuit is correct. Given enough time for the circuit to stabilise between consecutive computation cycles (which constitutes the synchronous design approach), this is indeed true, but it is obviously advantageous to present the circuit with new data as soon as the computation of previous data is complete. However, since no assumptions about gate delays are made in this approach, this can quickly lead to problems. For example, the following firing sequence:

$$\langle c+, a+, b+, g0+, g1+, c-, a-, b-, g1-, c+, g1+\rangle$$

leads the gate $g1$ into firing prematurely, which happens because the new wave of inputs arrives before gate $g0$ could return back into a stable state. This produces an incorrect behaviour of the circuit. If one tries to avoid this situation by substituting the second AND gate with a C-element (Figure 1 (b)) in order to synchronise the two gates, another problematic sequence surfaces:

$$\langle c+, a+, b+, g0+, g1+, c-\rangle$$

after which the output q will remain stable, even though one of the inputs is low, which is sufficient to state that this circuit is not a 3-input AND gate.

This illustrates the complex nature of interactions of the elements in an asynchronous circuit. Detection of all possible coincidences that may result in the incorrect behaviour of a circuit is therefore not a trivial task, and, considering that the modern technology requires to take into account not only possible delays of the logic gates, but also delays of the wires, it is also extremely computationally expensive. Several approaches are known that alleviate the state space explosion problem [23], [22], [12], most of them based on compressed representation of the reachability graph.

This paper presents an alternative, Petri net based approach to the problem of asynchronous circuit verification. To compress the state space, Petri net unfolding techniques are employed, which represent the state space implicitly.

III. Circuits and Petri nets

Petri nets [19] are a simple yet quite powerful formalism inherently suited for describing asynchronous systems. They

Authorized licensed use limited to: Newcastle University. Downloaded on May 26,2010 at 15:37:51 UTC from IEEE Xplore. Restrictions apply.

allow to model major aspects of behaviour of such systems, including concurrency, causality and conflict [30]. However, and also due to their simplicity, the size of the net required to model a system with complex behaviour can be very large, quickly becoming virtually unintelligible for the designer. To overcome this limitation, a designer can be presented with a higher-level view of a system, where the blocks of the underlying Petri net are represented as compact high-level objects, while the Petri net itself is used "behind-the-scenes" to perform verification of certain properties of the system [20], [26].

Definition 1: A Petri net (*PN*) is a quadruple $N = \langle P, T, F, m_0 \rangle$, where P is a finite non-empty set of places, T is a finite non-empty set of transitions, $F \subseteq (T \times P) \cup (P \times T)$ is the flow relation between places and transitions and m_0 is the initial marking. A pair $(p, t) \in F$ is called an arc. A Petri net marking is a function $m : P \to \mathbb{Z}_+$, where $m(p)$ is called the number of tokens in place $p \in P$ at the marking m. The set of places $\bullet t = \{p \in P \mid (p, t) \in F\}$ is called the preset of a transition $t \in T$, and $t\bullet = \{p \in P \mid (t, p) \in F\}$ is called the postset of t. A transition $t \in T$ is enabled at marking m if $\forall p \in \bullet t, m(p) > 0$. A transition $t \in T$ enabled at marking m can fire, producing a new marking m' (denoted $m[t\rangle m'$), such that

$$\begin{cases} m'(p) = m(p) - 1, p \in \bullet t \setminus t\bullet \\ m'(p) = m(p) + 1, p \in t \bullet \setminus \bullet t \\ m'(p) = m(p), p \in t \bullet \cap \bullet t \end{cases}$$

thus achieving the flow of information within the net.

A very useful extension of a plain Petri net is a labelled Petri net:

Definition 2: A labelled Petri net (*LPN*) is a 6-tuple $S = \langle P, T, F, m_0, \Sigma, \lambda \rangle$, where $\langle P, T, F, m_0 \rangle$ is a Petri net, Σ is a finite alphabet and λ is a function $\lambda : T \to \Sigma$ associating each transition of a Petri net with a label.

This allows for some meaningful semantics to be attached to the transitions. For example, each transition may be labelled with the name of an event. Then, having observed a sequence of transition firings, one can judge from that a sequence of events that happened in the system modelled by the Petri net. If a labelled Petri net is used to describe a system of switching binary signals, this leads to the notion of a signal transition graph:

Definition 3: A signal transition graph (STG) is a tuple $G = \langle P, T, F, m_0, \lambda, I, O, v_0 \rangle$, where $\langle P, T, F, m_0 \rangle$ is a PN, I is a set of input signals, O is a set of output signals, $I \cap O = \emptyset$, $v_0 = \{0, 1\}^{|Z|}$is a a vector of initial signal values, $Z = I \cup O = \{z_1, z_2, ..., z_{|Z|}\}$ is a joint set of all signals, λ is an injective labelling function $\lambda : T \to Z \times \{+, -\}$, i.e. an STG is an LPN where each transition is labelled with a signal level change event. If different transitions correspond to the same event, an index is used to distinguish them. Note that graphically, a signal event and its index are separated using a slash symbol, and if there is only one instance of the event, the index is omitted.

An idea to represent switching circuits as a special class of Petri nets was first proposed in [7] and further refined in [28]. For a long time, this approach was deemed inefficient due to the fact that several places and transitions, as well as a set of connecting arcs, are required to represent each signal (as opposed to a pair of Boolean equations used in BDD-based approaches [21]). However, in the light of recent developments in Petri net verification techniques, particularly of the tools based on unfolding theory [10] this approach cannot be ignored: the finite prefix of a Petri net unfolding is usually able to represent all of the possible behaviours of the net in a very compact way.

Definition 4: A circuit [21] is a triple $C = \langle V, \mathcal{F}, s_0 \rangle$, where $V = \{v, v_1, ..., v_n\}$ is a set of signals, $\mathcal{F}$ is a mapping $\mathcal{F} : v_i \to f_{v_i}, v_i \in V$ where f_{v_i}corresponds to the function of a logic gate that drives v_i and $s_0$is the initial state of the circuit.

Definition 5: A circuit Petri net R associated with a circuit C is a type of STG that satisfies the following properties:

1) For each signal $v_i \in V$ there exist exactly two complementary places $\{p_{v_i}, \overline{p_{v_i}}\} \in P$, such that at any reachable marking one and only one of $\{p_{v_i}, \overline{p_{v_i}}\}$ is marked with a single token. If in the initial state $s_0 \in C$ signal v_i is high, then at the initial marking $m_0 \in R$ place p_{v_i} is marked. Otherwise, $\overline{p_{v_i}}$ is marked.
2) For each pair of complementary places, there exists a finite number of positive transitions $t^+_{v_i} \in T$ that transfer the token from the place $\overline{p_{v_i}}$ to the place p_{v_i}, corresponding to the event of signal v_i going from low to high. Similarly, there exists a finite number of negative transitions $t^-_{v_i} \in T$ that transfer the token from p_{v_i} to $\overline{p_{v_i}}$, corresponding to the event of signal v_i going from high to low.
3) Transitions between complementary places are controlled by a set of read arcs that non-destructively test the presence of tokens in other places in P. The read arcs are placed in such a manner that they correspond to the dependence of a signal $v_i \in V$ on other signals in V exactly as defined by $\mathcal{F}(v_i)$.

As can be seen from definition 5, a circuit Petri net consists of a number of so-called *elementary cycles* interconnected with read arcs. A *read arc* [15] is an undirected arc connecting a place p with a transition t and interpreted in such a way that t can fire only when p is marked, however the firing of t does not remove any tokens from p. A read arc can be simulated in a usual Petri net by substituting it with a pair of arcs $\{\{p, t\}, \{t, p\}\}$ so that the token is put back to the place p after the firing of t which first consumes it. However, this approach leads to certain complications which will be discussed later.

An elementary cycle is a set of two complementary places and the transitions connecting them, associated with a signal in the circuit via labelling. Figure 2 shows the structure of such elementary cycles, and the way of producing different causality relations. In the figure, the places of the circuit Petri net are shown as circles (if a place is marked with a token, a dot is drawn inside), the transition as boxes, regular arcs as thin lines with arrowheads, and the read arcs as thick lines with no arrows; (a) is an elementary cycle with only one positive and one negative transition; (b) is an elementary cycle

Authorized licensed use limited to: Newcastle University. Downloaded on May 26,2010 at 15:37:51 UTC from IEEE Xplore. Restrictions apply.

with two positive transitions and one negative transition, which means that the signal it represents exhibits OR-causality for positive excitation and AND-causality for negative excitation (hence an OR-gate is driving it); (c) is an elementary cycle with OR-causality for negative excitation and AND-causality for positive excitation, which suggests that an AND-gate is driving the associated signal.

A. *Analysis of Petri nets*

Checking whether a Petri net satisfies a certain property is very important for the analysis of the system the net models. In particular, the notions of *marking reachability* and *deadlock-freedom* are used throughout this paper:

The set of reachable markings of a Petri net is the smallest (w.r.t. $\subseteq$) set $\mathcal{RM}$ containing m_0 and such that if $m \in \mathcal{RM}$ and $m[t\rangle m'$, for some $t \in T$ then $m' \in \mathcal{RM}$. A marking m is reachable if $m \in \mathcal{RM}$.

A marking m is deadlocked if at this marking no transitions are enabled. A Petri net is deadlock-free if none of its reachable markings is deadlocked.

To determine if there exists a reachable marking satisfying certain properties, the set of reachable markings $\mathcal{RM}$ must be computed. This, however, quickly leads to a combinatorial explosion problem, and requires state space compression techniques to be employed, such as BDDs [23] or Petri net unfoldings.

Given a Petri net N, the unfolding technique [10] aims at building a labelled acyclic net Unf_N (prefix) satisfying two key properties:

- Completeness. Each reachable marking of N is represented by at least one 'witness', i.e., one marking of Unf_N reachable from its initial marking. Similarly, for each possible firing of a transition at any reachable state of N there is a suitable 'witness' event in Unf_N.
- Finiteness. The prefix is finite and thus can be used as an input to model checking algorithms, e.g., those searching for deadlocks.

A prefix satisfying these two properties can be used for model checking as a condensed representation of the state space of a system. Since its introduction [14], the unfolding-based approach has been extensively improved and is able to deal with more complex and varied applications. In particular, recent research has shown that many verification problems for unfoldings can be formulated in terms of Boolean satisfiability (SAT) and very efficiently dealt with by existing SAT solvers [9].

IV. Construction of a circuit Petri net

Given a source gate-level model, a corresponding circuit Petri can be produced using Algorithm 1. However, several further steps are necessary before the Petri net can be fed to the external tools for verification. These steps are detailed below.

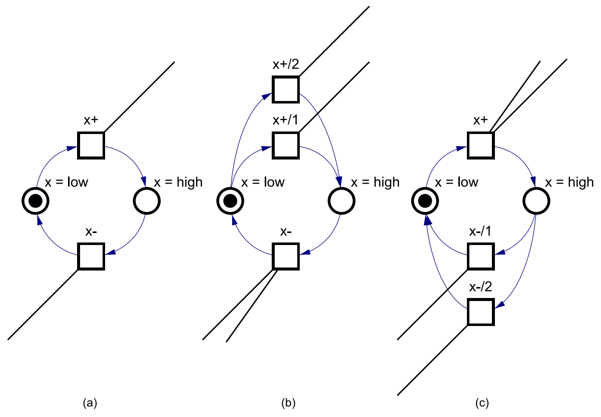
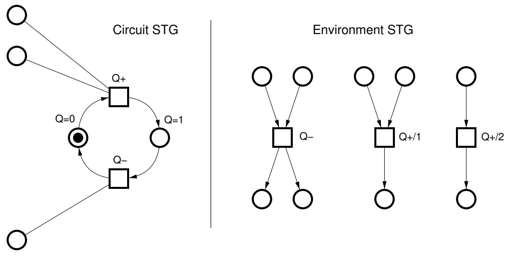


Figure 2. Examples of elementary cycles in circuit Petri net

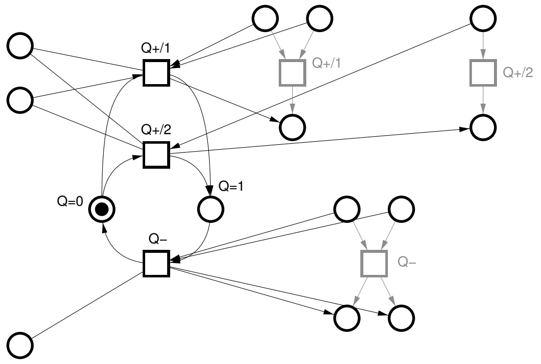
Algorithm 1 Conversion to circuit Petri net

```
for each signal $v_i \in V$:
  insert places $\{p_{v_i}, \overline{p_{v_i}}\}$into $P$
  if $v_i$ is high in $s_0$then
    mark $p_{v_i}$
  else
    mark $\overline{p_{v_i}}$
  end if
end for

for each signal $v_i \in V$:
  build a DNF $DNF_{set}$ for function $\mathcal{F}(v_i)$
  perform Boolean minimisation* of $DNF_{set}$
  k = 0
  for each clause C∈$DNF_{set}$:
    insert a transition $t_{v_i}^{+k}$into $T$
    insert arcs $\left\{(p_{v_i}, t_{v_i}^{+k}), (t_{v_i}^{+k}, \overline{p_{v_i}})\right\}$ into $F$
    for each signal $v_j \in C$:
      if $v_j$ is negated then
         insert arcs $\left\{(\overline{p_{v_j}}, t_{v_i}^{+k}), (t_{v_i}^{+k}, \overline{p_{v_j}})\right\}$ into $F$
      else
         insert arcs $\left\{(p_{v_j}, t_{v_i}^{+k}), (t_{v_i}^{+}, p_{v_j})\right\}$ into $F$
      end if
    end for
    increment k
 end for
   build a DNF $DNF_{reset}$ for function $\overline{\mathcal{F}(v_i)}$
   perform Boolean minimisation of $DNF_{reset}$
   k = 0
   for each clause C∈$DNF_{reset}$:
     insert a transition $t_{v_i}^{-k}$into $T$
     insert arcs $\left\{(\overline{p_{v_i}}, t_{v_i}^{-k}), (t_{v_i}^{-k}, p_{v_i})\right\}$ into $F$
     for each signal $v_j \in C$:
      if $v_j$ is negated then
        insert arcs $\left\{(\overline{p_{v_j}}, t_{v_i}^{-k}), (t_{v_i}^{-k}, \overline{p_{v_j}})\right\}$ into $F$
      else
        insert arcs $\left\{(p_{v_j}, t_{v_i}^{-k}), (t_{v_i}^{-k}, p_{v_j})\right\}$ into $F$
      end if
     end for
     increment k
   end for
 end for

*using Quine-McCluskey algorithm [13]
```

Authorized licensed use limited to: Newcastle University. Downloaded on May 26,2010 at 15:37:51 UTC from IEEE Xplore. Restrictions apply.



(a) Circuit and environment STGs



(b) Compositional STG

Figure 3. Composition of circuit and environment STGs

A. Applying environment interface

After the circuit Petri net has been constructed, it is necessary to compose it with the provided environment interface STG. This is done by superposition of the corresponding transitions of the two Petri nets. Figure 3 shows an example of such superposition of transitions corresponding to the output signal Q. In the circuit Petri net, there is a positive transition $Q+$ and a negative transition $Q-$. Environment STG contains two occurrences of positive transition$\{Q+/1, Q+/2\}$ and one negative transition $Q-$ (see Subfigure(a)). The superposition of $Q-$ transition is trivial: it is removed from environment STG and the token flow is redirected through $Q-$ transition in the circuit Petri net. This is not possible with the positive transition $Q+$: it needs to be duplicated in the circuit Petri net to create two transitions $\{Q+/1, Q+/2\}$ with the same preset and postset. After that these two transitions can be superpositioned with the corresponding transitions in the environment STG (see Subfigure(b)).

B. Read arcs complexity reduction

It should be noted that the available Petri net unfolding tools do not recognise read arcs as a special type of arc. Instead, read arcs need to be modelled as double-headed arcs, i.e. if $p \in \bullet t \cap t\bullet, p \in P, t \in T$ then p and t are connected with a read arc. Though behaviourally correct, this representation is semantically different from an actual read arc in that it introduces a choice, which may lead to a drastic growth of the unfolding size. This problem can be resolved to an extent by ensuring that any place is associated with at most one read arc [29], which can be accomplished by making a necessary number of copies of each place with multiple outgoing read arcs and rearranging the read arcs accordingly, as shown in (Figure 4).

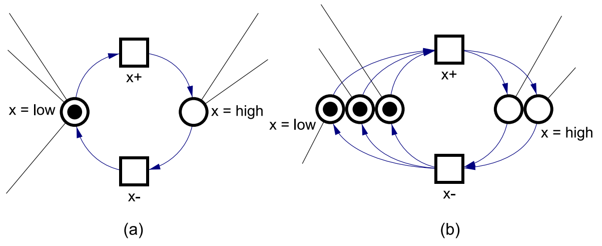


Figure 4. Read arcs complexity reduction
(a) multiple read arcs associated with one place
(b) only one read arc per place

V. Verification

A circuit is considered speed-independent under a given environment, if

1) It conforms to the environment, i.e. produces only those changes of output signals that do not conflict with the environment's STG.
2) It is hazard-free.

In the scope of this paper, a hazard is defined to be an unexpected change of the input signal of a gate, such that it causes an enabled (positively or negatively excited) gate to become disabled (i.e. to return into a stable state without firing). A circuit that never exhibits such behaviour is called *hazard-free,* or *safe.*

A. Detection of potential hazards

A pair of signals is called *conflicting* if there exists a reachable state of the circuit such that a change in the level of one of them disables the gate driving the other. In terms of a circuit Petri net, a potentially hazardous state is a state which violates the *semi-modularity* property:

Definition 6: A Petri net is called semi-modular if any transition in this net, once enabled, cannot be disabled until it has fired.

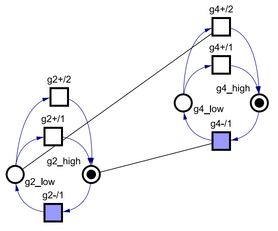
In other words, once each place in the preset of a transition has become marked with a token, thus enabling the transition, no other transition can "steal" any of these tokens. In Figure 5 (a), an example of non-semi-modularity is shown: if transition $g2-/1$ fires, it disables transition $g4-/1$ (enabled transitions are depicted as greyed boxes).

Definition 7: A pair of transitions $\{t_1, t_2\} \in T$ is called *conflicting* if $\bullet t_1 \cap \bullet t_2 \neq \emptyset$.

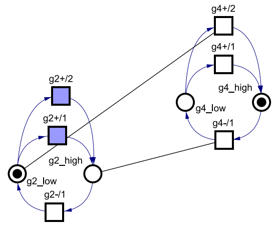
For the purpose of verification, we consider that if a circuit Petri net is semi-modular, then the circuit it was constructed from by using Algorithm 1 is hazard-free.

This statement stems from the following: for each signal in the circuit, there is an elementary cycle (see Section III) in the Petri net, and for each possible combination of the levels of input signals which lead the gate that drives this signal into a positively or negatively excited state, there is a corresponding positive or negative transition in this cycle. Once any of these

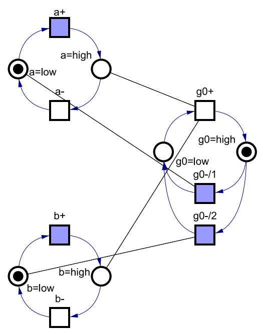
Authorized licensed use limited to: Newcastle University. Downloaded on May 26,2010 at 15:37:51 UTC from IEEE Xplore. Restrictions apply.



(a) Conflicting transitions



(b) Transitions in allowable conflict



(c) Transitions in allowable conflict

Figure 5. Non-semi-modular states

combinations becomes active (the gate becomes excited), the corresponding transition becomes enabled. If the state of any of the input signals changes in such a way that the excitation condition is no longer fulfilled and the gate has not yet fired, this produces hazard, but it will also cause the corresponding circuit Petri net transition to become disabled, thus violating semi-modularity (see also [16] for Muller's original view of semi-modularity).

However, while the presence of a potential hazard in the source gate-level model will always indicate a violation of the semi-modularity in the circuit Petri net, the reverse is not true. There are two cases in which a violation of semi-modularity in the circuit Petri net does not indicate the presence of a potential hazard in the original circuit.

The first situation arises due to the possibility of several transitions representing the same signal event, but being caused by different preceding events, as shown in Figure 5. In Subfigure (b), the conflicting transitions $g2+/1$ and $g2+/2$ represent the same event, signal $g2$ going high. Hence, the conflict of these transitions does not constitute a signal conflict: they both have the same semantics and thus their firing does not disable any other signal. In Subfigure (c), the conflict between $g0-/1$ and $g0-/2$ is allowed for the same reason, but there is also conflict between $a/+$ and $g0-/1$ which are associated with different signals. However, even if $a/+$fires, disabling $g0-/1$, the enabled transition $g0-/2$ still keeps the signal event $g0-$ enabled, and thus disabling of the transition $g0-/1$ does not lead to the disabling of the negatively excited gate driving signal $g0$, so there is again no signal conflict. On the other hand, if the transition $g0-/2$ was not enabled, then the conflict between $a/+$ and $g0-/1$ would be a signal conflict.

The second situation occurs when the conflicting transitions are both associated with the input signals. Since it is the environment that controls these signals, this situation should be considered a choice of mode of circuit operation made by the environment and not a signal conflict.

To summarise, if for all conflicting pairs of transitions $\{t_1, t_2\} \in T$:

1) $\lambda(t_1)$ and $\lambda(t_2)$ are not both input signal events
2) $\lambda(t_1) \neq \lambda(t_2)$
3) there exists a reachable marking such that $\forall p \in \bullet t_1 \cup \bullet t_2, m(p) > 0$ and at this marking there is no enabled transition $t \in T$ such that $(\lambda(t) = \lambda(t_1)) \vee (\lambda(t) = \lambda(t_2))$

then there is a potential hazard in the original circuit.

B. Detection of interface non-conformance

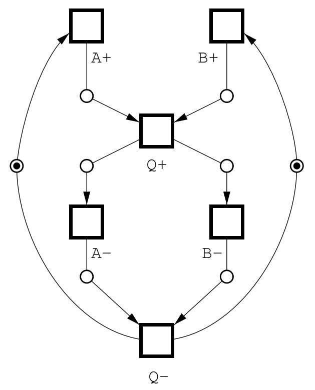
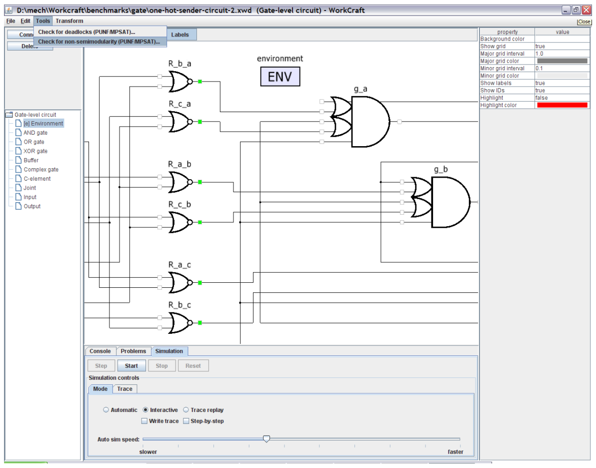


Figure 6. A C-element interface STG

A circuit Petri net, when composed with its environment, forms a closed system: the outputs of the circuit are the inputs for the environment STG, and vice versa. Thus, the conformance verification is twofold: if the environment part of the composed Petri net is able to produce a sequence of inputs that causes "bad behaviour" of the circuit (i.e. a hazard or a deadlock), the circuit is said not to conform to its environment and this situation is referred to as α*-non-conformance*; on the other hand, if the circuit is ever able to produce an output signal change that is not expected by the environment, it is also said not to conform to the environment, and this situation is referred to as β*-non-conformance.*

An example of α-non-conformance can be demonstrated if a XOR gate is verified against the C-element interface STG (Figure 6), i.e. by verifying whether a XOR-gate properly implements the C-element STG. If the environment produces events $A+$ and $B+$ almost simultaneously, quickly enough so that the XOR gate becomes excited but does not fire and returns into stable (output signal low) state, this leads to, first, a hazard on one of the inputs, and, second, into a deadlock. The deadlock is present because the C-element environment, having switched both input signals to high, expects an output signal Q to go high. But this never happens: a XOR gate cannot switch output signal to high until one of its input

Authorized licensed use limited to: Newcastle University. Downloaded on May 26,2010 at 15:37:51 UTC from IEEE Xplore. Restrictions apply.



property	value
Background color	
Show grid	true
Major grid interval	1.0
Major grid color	▇
Minor grid interval	0.1
Minor grid color	
Show labels	true
Show IDs	true
Highlight	false
Highlight color	🟥

Figure 7. Workcraft framework UI

signals goes low, and this will never happen as well, because the STG does not allow to reset the inputs A and B until the output Q is produced. Thus, α-non-conformance is decided by checking the Petri net for hazards and deadlocks. A method for hazard detection is explained in Subsection V-A, and the deadlock problem is solved by external model-checking tools, thus checking for α-non-conformance does not require much additional effort.

If the XOR gate is replaced by an AND gate, however, there is no α-non-conformance: the input goes high only when both outputs go high, thus no hazard is observed. But when either one of the inputs goes low, the AND gate becomes negatively excited, and tries to reset the output, which is not expected by the environment STG. However, in the corresponding compositional Petri net the environment *restricts* the circuit because the two transitions $Q-$(one provided by the circuit, and the other by the environment) become superimposed (Subsection IV-A), which introduces a synchronisation, and thus the transition $Q-$ will only become enabled when the environment resets the second input signal. Hence, the system has no hazards and no deadlock, but the AND gate obviously does not conform to the C-element interface. If it would have not been restricted by environment, it could produce event $Q-$ when it was unexpected, exhibiting β-non-conformance.

Let C be a circuit, R be a circuit Petri net constructed from C and E be the environment STG. Let $R.P$ denote the set of places $P \in R$, $E.P$ the set $P \in E$ and M the set of all transitions which were superimposed during circuit-environment composition. Then, if there exists a reachable marking m such that at this marking for at least one transition from M, all of the places in its preset that belong to R are marked, and there exists at least one place in its preset that belongs to E which is not marked, or, formally, $\exists t \in M : (\forall p \in \bullet t \cap R.P, m(p) > 0) \wedge (\exists p \in \bullet t \cap E.P, m(p) = 0)$ then the circuit C is β-non-conformant under environment E.

VI. Workcraft framework

The visual editing of gate-level models, their simulation, conversion into circuit Petri nets and verification by external model checking tools - these steps are done automatically and integrated in a consistent framework called Workcraft (Figure 7). The framework has a plug-in driven architecture and supports run-time scripting, which makes it a flexible and expandable environment. Its underlying Java technology provides robust cross-platform operation. Workcraft uses OpenGL hardware acceleration for real-time visualisation, which allows interactive animated simulation of large specifications. The framework also provides a transparent interface to model checking tools, such as PUNF and MPSAT, for seamless verification of the high-level models, such as SDFS [26] using their low-level Petri net representation.

All the circuits presented in this paper were created and analysed using the Workcraft framework. The capabilities of Workcraft are not limited to circuit models: there are plug-ins that support editing, simulation and analysis of generic graphs, marked graphs, Petri nets, unfoldings, SDFS models [26] and others. As a useful side-feature, there is also a vector graphics export plug-in: most of the figures in this paper were imported directly from Workcraft with minimal modification.

For more information on this tool, please refer to [20].

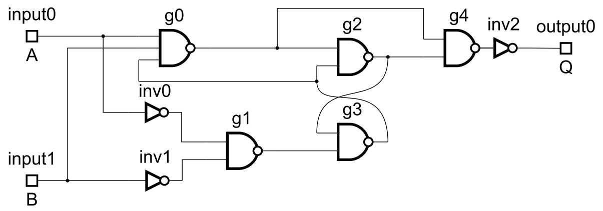


Figure 8. NAND C-element implementation

This section presents an example of application of the method proposed above and implemented in the Workcraft tool, and demonstrates the achieved integrity of the design workflow. Figure 8 shows a NAND-based implementation of the C-element proposed by Maevsky. The gate-level model was created using Workcraft's visual editor and verified. The verification fails and reports a following trace as the shortest firing sequence that leads to a potential hazard:

$$\langle input1, input0, inv1, g0, g1, g2, g3, g0, g4, inv2, output0\rangle$$

The faulty trace can be simulated and the problematic firing sequence examined, which reveals that indeed, provided the $inv0$ inverter's delay is long enough, it can be excited but still not have fired after the environment has received output signal Q and resets input signals A and B, disabling $inv0$. However this is very unlikely, because in order for this hazard to actually happen, $inv0$ delay should be longer that the total delay of all other gates. Hence, this potential failure can be safely ignored for any practical application.

By replacing the inverters $\{inv0, inv1\}$ and the NAND gate $g1$ with an OR gate (Figure 9), this problem is eliminated and the verification reports success, confirming that the implementation of the C-element shown in Figure 9 is speed-independent, and the implementation shown of Figure 8 is speed-independent under a very reasonable timing assumption.

Authorized licensed use limited to: Newcastle University. Downloaded on May 26,2010 at 15:37:51 UTC from IEEE Xplore. Restrictions apply.

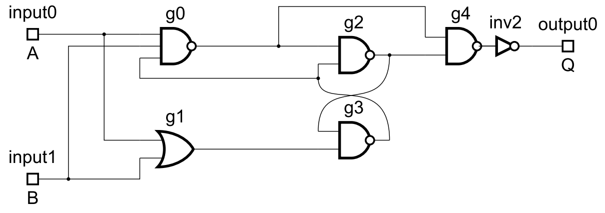


Figure 9. NAND-OR C-element implementation(no wire delays)

But while this circuit is speed-independent, it could still produce unexpected behaviour if it is not delay-insensitive. To verify whether it is delay-insensitive, possible wire delays should be taken into account. Since it is enough to demonstrate that delay on any of the wires may lead to a hazard in order to assert that the circuit is not delay-insensitive, it may be reasonable not to model delays on all of the wires in order to minimise verification time. In Figure 10, a wire delay is introduced in the form of a buffer into the fork following gate $g3$ output. Verification fails with the following trace:

$$\langle input1, input0, g1, g0, g2, g3, g0 \rangle$$

Examination of this trace shows that the hazard can happen if gate $g0$, after receiving the signal from $g3$, will switch before the same signal from $g3$, but travelling across the other branch of the fork, reaches gate $g2$. In this case, the firing of $g0$ will disable the already excited gate $g2$. This is enough to state that this C-element implementation is not strictly delay-insensitive, but requires a timing assumption that the delay of signal reaching $g0$ plus $g0$ switching delay is more than wire delay on the other branch of the fork.

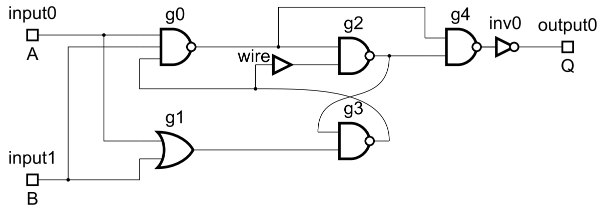


Figure 10. NAND-OR C-element implementation
(wire delay present on one fork only)

It may still be helpful to check this circuit considering all the possible wire delays. This can be done by providing branches of all forks with buffers (the buffers are not needed on non-branching wires, and on the sections of wire preceding forks, because in this case it may simply be considered that the delay of the gate producing signal on this wire includes the wire delay), as shown in Figure 11. Verification in this case produces the following failure trace:

$$\langle input1, input0, w2, w0, g0, w7, g2 \rangle$$

This failure is similar to the one in the case above: if the delay of $w7 + g2$ is less than delay of $w6$, $g4$ will be disabled before it can fire. Note that the verification time is considerably longer due to the growth of unfolding prefix.

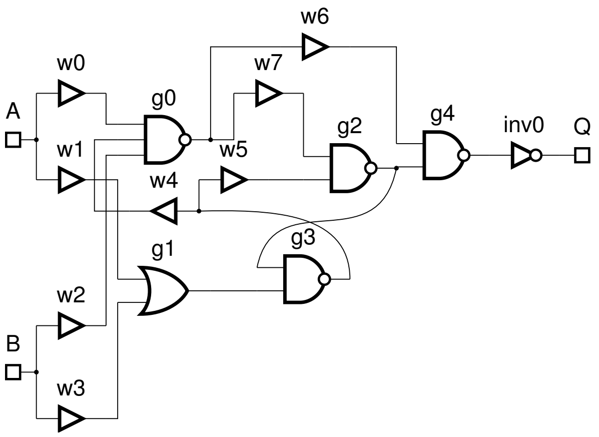


Figure 11. NAND-OR C-element implementation(full set of wire delays)

VII. Performance and comparison statistics

The presented verification approach was tested on a set of benchmarks (see Table I) which included asynchronous multiport registers [17] and FIFO pipelines [11].

The results are compared with Versify [21] and zeta [12] tools. Note: the runtimes for Versify were taken from [21] and thus cannot be compared directly with the results for zeta and Workcraft because the latter were obtained on a modern machine. The times for Versify are provided in order to highlight the rapid growth of the runtime due to exponential growth of the state space. It is possible to see that Versify and zeta runtimes grow considerably faster with the growth of the number of states that the size of the unfoldings and reachability analysis time, which in many cases grow linearly because the analysed circuits exhibit high degree of concurrency.

It should be also mentioned that Petri net unfolding algorithm is parallelisable [8] thus its runtime can be greatly reduced on a multiprocessor system whilst computations for BDD-based techniques cannot be easily distributed between multiple processing units.

VIII. Incorporation of timing

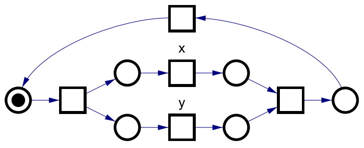
Looking at the examples from section 7, one could notice that the states that lead to a potential hazard can be avoided by introducing timing assumptions. *Timing assumption* is an assertion that a certain event in the system will always happen before certain other event. For the example shown in Figure 8, the statement "inverter $inv0$ will switch before an output is generated" could be such an assumption, and if it had been taken into consideration during verification, the reported dangerous state would not have been reachable. If a timing assumption is available, it can be easily introduced into the circuit Petri net by employing concurrency reduction (Figure 12). In the system shown in Subfigure (a), events x and y can happen in any order; in the system in Subfigure (b), event y is always preceded by event x, and in the system in Subfigure (c), event x is always preceded by event y.

Specifying such limitations manually on the gate-level model is a straightforward, but not very practical way, because

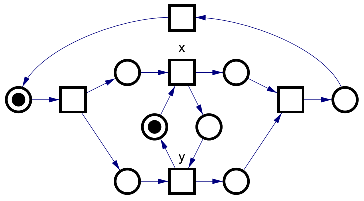
Authorized licensed use limited to: Newcastle University. Downloaded on May 26,2010 at 15:37:51 UTC from IEEE Xplore. Restrictions apply.

Benchmark	States	Net size (P/T)	Unfolding size (events/cutoffs)	VERSIFY	ZETA	WORKCRAFT (PUNF+MPSAT)
reg2	$2.5*10^{4}$	183/124	368/29	n/a	0.47 sec	0.11 sec
reg4	$7.6*10^{7}$	337/220	2464/177	388 sec	2.75 sec	6.33 sec
reg8	$7.1*10^{14}$	649/416	72192/4865	7246 sec	83.9 sec	48.38 sec
fifo5	$2.6*10^{3}$	97/58	86/1	8 sec	0.15 sec	0.02 sec
fifo10	$1.2*10^{6}$	177/108	166/1	130 sec	0.61 sec	1.02 sec
fifo15	$5.8*10^{8}$	257/158	246/1	634 sec	3.99 sec	2.4 sec

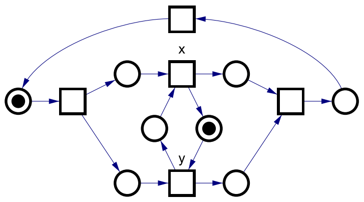
Table I
COMPARISON OF PROPOSED METHOD WITH EXISTING TOOLS



(a) x is concurrent with y



(b) y always follows x



(c) x always follows y

Figure 12. Concurrency reduction

for a large enough system there can exist a multitude of valid timing assumptions. In addition, manually specified timing information may become obsolete due to technology scaling. To deal with these issues, an automated technique is required, which would extract timing information from the gate libraries and generate all possible timing assumptions accordingly. This is the focus of the future work.

IX. CONCLUSION

An algorithm for automatic conversion of a circuit netlist into a behaviourally equivalent Petri net and the method for composing of the resultant net with the environment specification is proposed. An approach to verification of the compositional net based on model-checking is discussed, and several examples are studied. Experimental results and comparison with previously existing tools are demonstrated. The direction of the future research is discussed.

Acknowledgements. The authors would like to thank Victor Khomenko for many fruitful discussions regarding Petri net unfoldings, and for his help with PUNF and MPSAT tools, Tomohiro Yoneda for the quick response and valuable assistance with the zeta tool and Oriol Roig for the help with Versify.

REFERENCES

[1] Manoj Ampalam and Montek Singh. Counterflow pipelining: architectural support for preemption in asynchronous systems using anti-tokens. In *Proc. International Conference Computer-Aided Design (ICCAD)*, November 2006.

[2] Peter A. Beerel, Jerry Burch, and Teresa H.-Y. Meng. Efficient verification of determinate speed-independent circuits. In *ICCAD '93: Proceedings of the 1993 IEEE/ACM international conference on Computer-aided design*, pages 261–267, Los Alamitos, CA, USA, 1993. IEEE Computer Society Press.

[3] Charles Brej. *Early output logic and anti-tokens*. PhD thesis, Dept. of Computer Science, University of Manchester, 2005.

[4] Thomas Chatain. *Symbolic Unfoldings of High-Level Petri Nets and Application to Supervision of Distributed Systems*. PhD thesis, Universite de Rennes 1, 2006.

[5] Tam-Ahn Chu. *Synthesis of self-timed VLSI circuits from graph-theoretic specifications*. PhD thesis, 1987.

[6] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. *IEICE Transactions on Information and Systems*, E80-D(3):315–325, 1997.

[7] J. Grabowski. On the analysis of switching circuits by means of petri nets. *Elektronische Informations-verarbeitung und Kybernetik*, 14:611–617, 1978.

[8] Keijo Heljanko, Victor Khomenko, and Maciej Koutny. Parallelisation of the petri net unfolding algorithm. In *Tools and Algorithms for Construction and Analysis of Systems*, pages 371–385, 2002.

[9] V. Khomenko. Computing shortest violation traces in model checking based on petri net unfoldings and sat (technical report cs-tr-84). Technical report, School of Computing Science, Newcastle University, 2004.

[10] Victor Khomenko. *Model Checking Based on Prefixes of Petri Net Unfoldings*. PhD thesis, University of Newcastle upon Tyne, School of Computing Science, 2003.

[11] Alain J. Martin. Self-timed FIFO: an exercise in compiling programs into VLSI circuits. *HDL description to guaranteed correct circuit design, North-Holland*, 1986.

[12] Koichi Masukura, Moniru Tomisaka, and Tomohiro Yoneda. Verification of asynchronous circuits based on zero-suppressed BDDs. *Systems and Computers in Japan*, 32:43–54, 2001.

[13] E. J. McCluskey. Minimization of Boolean functions. *Bell Syst. Tech.*, 35:1417–1444, 1956.

[14] Kenneth L. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem*. PhD thesis, Carnegie-Mellon University Pittsburgh Dept. of Computer Science, 1992.

[15] U. Montanari and F. Rossi. Contextual nets. *Acta Informacia*, 32(6):545–596, 1995.

[16] D. E. Muller and W. C. Bartky. A theory of asynchronous circuits. *Annals of Computing Laboratory of Harvard University*, pages 204–243, 1959.

[17] Takashi Nanya, Yoichiro Ueno, Hiroto Kagotani, Masashi Kuwako, and Akihiro Takamura. Titac: Design of a quasi-delay-insensitive microprocessor. *IEEE Des. Test*, 11(2):50–63, 1994.

[18] Curtis A. Nelson, Chris J. Myers, and Tomohiro Yoneda. Efficient verification of hazard-freedom in gate-level timed asynchronous circuits. In *ICCAD '03: Proceedings of the 2003 IEEE/ACM international conference on Computer-aided design*, page 424, Washington, DC, USA, 2003. IEEE Computer Society.

Authorized licensed use limited to: Newcastle University. Downloaded on May 26,2010 at 15:37:51 UTC from IEEE Xplore. Restrictions apply.

[19] Carl Adam Petri. *Kommunikation mit Automaten.* PhD thesis, Bonn: Institut für Instrumentelle Mathematik, Schriften des IIM Nr. 2, 1962. Second Edition:, New York: Griffiss Air Force Base, Technical Report RADC-TR-65–377, Vol.1, 1966, Pages: Suppl. 1, English translation.

[20] Ivan Poliakov, Danil Sokolov, and Andrey Mokhov. Workcraft: A static data flow structure editing, visualisation and analysis tool. In *Petri Nets and Other Models of Concurrency - ICATPN 2007*, 2007.

[21] Oriol Roig. *Formal Verification and Testing of Asynchronous Circuits*. PhD thesis, Universitat Politecnica de Catalunya, 1997.

[22] Oriol Roig, Jordi Cortadella, and Enric Pastor. Hierarchical gate-level verification of speed-independent circuits. In *Asynchronous Design Methodologies*, pages 129–137. IEEE Computer Society Press, 1995.

[23] Oriol Roig, Jordi Cortadella, and Enric Pastor. Verification of asynchronous circuits by BDD-based model checking of Petri nets. In *16th International Conference on the Application and Theory of Petri Nets*, volume 815, pages 374–391, 1995.

[24] Leonid Rosenblum and Alex Yakovlev. Signal graphs: from self-timed to timed ones. In *International Workshop on Timed Petri Nets*, pages 199–207. IEEE Computer Society Press, 1985.

[25] Sanjit A. Seshia, Randal E. Bryant, and Kenneth S. Stevens. Modeling and verifying circuits using generalized relative timing. In *ASYNC '05: Proceedings of the 11th IEEE International Symposium on Asynchronous Circuits and Systems*, pages 98–108, Washington, DC, USA, 2005. IEEE Computer Society.

[26] Danil Sokolov, Ivan Poliakov, and Alex Yakovlev. Asynchronous data path models. In *7th International Conference on Application of Concurrency to System Design*, 2007.

[27] Jens Sparsø and Steve Furber, editors. *Principles of Asynchronous Circuit Design: A Systems Perspective*. 2001.

[28] V. Varshavsky, M. Kishinevsky, V. Marakhovsky, V. Peschansky, L. Rosenblum, A. Taubin, and B. Tzirlin. *Self-Timed Control of Concurrent Processes*. Kluwer Academic Publisher, Dordrecht, The Netherlands, 1990.

[29] Walter Vogler, Alexei L. Semenov, and Alexandre Yakovlev. Unfolding and finite prefix for nets with read arcs. In *International Conference on Concurrency Theory*, pages 501–516, 1998.

[30] A. Yakovlev, L. Lavagno, and A. Sangiovanni-Vincentelli. A unified signal transition graph model for asynchronous control circuit synthesis. In *ICCAD'92*, 1992.

[31] Hao Zheng, Chris J. Myers, David Walter, Scott Little, and Tomohiro Yoneda. Verification of timed circuits with failure directed abstractions. In *ICCD '03: Proceedings of the 21st International Conference on Computer Design*, page 28, Washington, DC, USA, 2003. IEEE Computer Society.

Authorized licensed use limited to: Newcastle University. Downloaded on May 26,2010 at 15:37:51 UTC from IEEE Xplore. Restrictions apply.