Encoding Large Asynchronous Controllers With ILP Techniques

Josep Carmona and Jordi Cortadella, *Member, IEEE*

***Abstract*—State encoding is one of the most difficult problems in the synthesis of asynchronous controllers. This paper presents a method that can solve the problem of large controllers specified with signal transition graphs. The method is based on the structural theory of Petri nets and uses integer-linear programming to insert state signals in locations that guarantee the consistency and absence of critical races. The structural nature of the proposed method makes it conservative, i.e., a solution cannot be guaranteed, even if it exists. Nevertheless, the experiments show that this limitation did not preclude finding a solution for all the examples presented in this paper. The method can be customized for area or delay optimization. The experimental results confirm the quality of the circuits, as compared with state-based methods. They also show the significant benefits that could be obtained if logic synthesis would be incorporated in synthesis frameworks that generate controllers by syntax-directed translation.**

***Index Terms*—Asynchronous circuits, Petri net (PN), state encoding, structural methods.**

I. Introduction

THE AUTOMATIC synthesis of asynchronous controllers has often faced a difficult and challenging problem, i.e., state encoding. The absence of critical races in the behavior of clockless circuits with high concurrence is a property that is not easy to meet. The strategies that have been proposed for several decades are intrinsically related to the circuit's mode of operation.

In the fundamental mode of operation, based on the work of Huffman [15] and Unger [33], the circuit and the environment sequentially work. Every burst of events produced by the environment at the inputs is followed by a burst of events produced by the circuit at the outputs. However, inputs and outputs never simultaneously change. These circuits are usually specified using flow tables, in which the rows represent states, and the columns represent signals. The encoding problem aims at producing a flow table, in which the input changes produce a set of internal transitions that, after visiting unstable states, always finish in terminal stable states [23], [32].

Manuscript received December 21, 2006; revised May 29, 2007. This work was supported in part by the Comisión Interministerial de Ciencia y Tecnología (CICYT) under Contract TIN2004-07925 and in part by the Generalitat de Catalunya through a Distinction for Research Fund. This paper is an extended version of [3]. This paper was recommended by Associate Editor S. Nowick.

The authors are with the Department of Software, Universitat Politècnica de Catalunya, 08034 Barcelona, Spain (e-mail: jcarmona@lsi.upc.edu; jordi.cortadella@upc.edu).

Digital Object Identifier 10.1109/TCAD.2007.907238

Another mode of operation is the burst mode [27], in which the environment is allowed to produce different sets of input bursts, with an associated output burst for each one. An extension has also been proposed to allow some restricted concurrence between inputs and outputs [38]. In [12], the problem was solved by adding specific constraints to the input encoding problem [25]. The additional constraints are the ones that guarantee the absence of critical races in the encoding.

The state encoding problem becomes even more difficult for the input/output mode of operation, in which no constraints on the concurrence between inputs and outputs are imposed. The signal transition graph (STG) [6], [29] is the most popular specification formalism for this type of circuits. This is the formalism used in this paper.

A. State Encoding for STGs

In [36], state encoding was formulated as a set of Boolean constraints to find a state variable assignment with race-free behavior. Solutions could be found by using a satisfiability (SAT) solver, but only very small specifications could be handled. In [14], the specification was first decomposed to reduce the complexity of the SAT problem proposed in [36]. However, the approach could only find suboptimal solutions for some subclasses of STGs.

In [19], the problem was solved by constructing a graph of encoding conflicts and then coloring the graph with binary encoded colors. This method was limited to specifications without choices.

A general method for state encoding was proposed in [7], which is based on the theory of regions of Petri nets (PNs). The insertion of new signals was restricted to regions and intersections of regions, thus guaranteeing a race-free encoding.

In all the previous methods, the set of reachable states must be generated to solve the encoding problem. Even though the method proposed in [7] used binary decision diagrams to symbolically represent sets of states, the applicability was reduced to specifications with not more than 20 signals. The state explosion problem has been the main reason that the previous methods have not been able to synthesize large controllers.

Recently, an approach for state encoding that combines partial order techniques with SAT has been presented as an alternative to state-based methods [17]. The underlying idea for a consistent signal insertion as the resolution of an SAT problem inspired the technique presented in this paper. The method in [17] is exact and requires a larger computational complexity than the conservative method presented in this paper.

B. Structural Methods

To overcome state explosion, some approaches have tackled the encoding problem without generating the state space. However, they have always been restricted to certain classes of PNs.

The simplest subclass is marked graphs that can specify choice-free behaviors. The approach proposed in [35] for marked graphs introduced the notion of lock relations to identify sets of complementary sequences and break them with the insertion of new signals.

The approach presented in [28] could handle free-choice PNs, taking advantage of the property that this subclass of specifications can be decomposed into state machine components. Similarly, Lin *et al.* [22] proposed an approach based on the decomposability of free-choice nets into marked graphs.

C. Contributions

This paper proposes a novel method for solving the state encoding problem that is not restricted to any subclass of PNs. It is based on the structural theory of PNs and uses the marking equation to conservatively characterize the reachability space of the system. By using integer-linear programming (ILP) models, it is possible to find insertion points for new state signals that guarantee the consistency and absence of critical races. Structural methods for detecting encoding conflicts and synthesizing speed-independent circuits have been presented in [2]. These methods transform the problems of detecting encoding conflicts and calculating support signals for synthesis to the feasibility of an ILP model. The missing part was a method for encoding the specifications, which is the contribution presented in this paper, and therefore, together with previous work on the structural detection of encoding problems and synthesis [2], a complete framework for the structural synthesis of asynchronous controllers can now be envisioned.

One of the main interests of the proposed approach is the capability of handling large STGs that are generated from hardware description languages (HDLs) such as Tangram [1] and Balsa [10]. Moreover, the approach presented in this paper can be combined with recent approaches for decomposition [30], widening the capabilities of applying logic synthesis in a complete synthesis flow. In fact, the main advantage of these HDLs is that they are supported by complete synthesis flows using the so-called syntax-directed translation (SDT) paradigm. The state encoding problem is avoided by implicitly overencoding the system with a netlist of handshake components that implement the parse tree of the specification. However, SDT does not benefit from the power of logic synthesis that manipulates the Boolean equations of the next-state functions, thus leading to unoptimized implementations.

In [4], a back end to incorporate logic synthesis into the Balsa system was presented. The work showed the tangible improvements that can be obtained by optimizing the netlists of handshake circuits. However, the underlying formalism for the synthesis are burst-mode machines, which impose limitations on modeling the inherent concurrence of asynchronous systems.

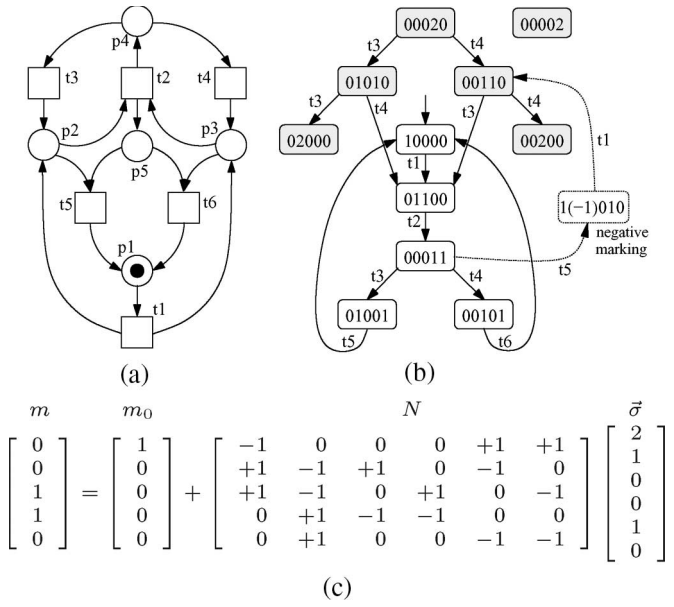


Fig. 1. (a) PN. (b) Potential reachability graph. (c) Marking equation.

This paper is organized as follows. Section II reviews the theory of PNs required for this paper and gives an overview of the state encoding approach by means of a simple example. The core of this paper is presented in Sections III and IV, where the main algorithm for signal insertion is described. Section V presents experimental results that validate the applicability of the encoding method.

II. Basic Theory and Introductory Example

The theory required in this paper is introduced by showing how the encoding problem of a simple VME bus controller can be solved.

A. PNs and STGs

A PN [26] is a four-tuple $N = \langle P, T, \mathcal{F}, m_0 \rangle$, where P is the set of places, T is the set of transitions, $\mathcal{F} : (P \times T) \cup (T \times P) \rightarrow \{0, 1\}$ is the flow relation, and m_0 is the initial marking. A PN is marked by assigning a nonnegative integer to each place. If k is assigned to place p by marking m, which is denoted by $m(p) = k$, we say that p is marked with k tokens. Given a node $x \in \mathcal{P} \cup \mathcal{T}$, its preset and postset are denoted by ${}^\bullet x$ and $x^\bullet$, respectively. An example of a PN is shown in Fig. 1(a).

A transition t is enabled in a marking m when all places in ${}^\bullet t$ are marked. When a transition t is enabled, it can fire by removing a token from each place in ${}^\bullet t$ and putting a token to each place in $t^\bullet$. A marking m' is reachable from m if there is a sequence of firings $t_1, t_2, \ldots, t_n$ that transforms m into m', which is denoted by $m[t_1, t_2, \ldots, t_n\rangle m'$. A sequence of transitions $t_1, t_2, \ldots, t_n$ is feasible if it is firable from m_0. The set of reachable markings from m_0 is denoted by $[m_0\rangle$ and forms a graph called the reachability graph. A PN is k bounded if no marking in $[m_0\rangle$ assigns more than k tokens to any place. A place p in a PN is implicit if the removal p and its incident arcs do not affect the behavior of the system.

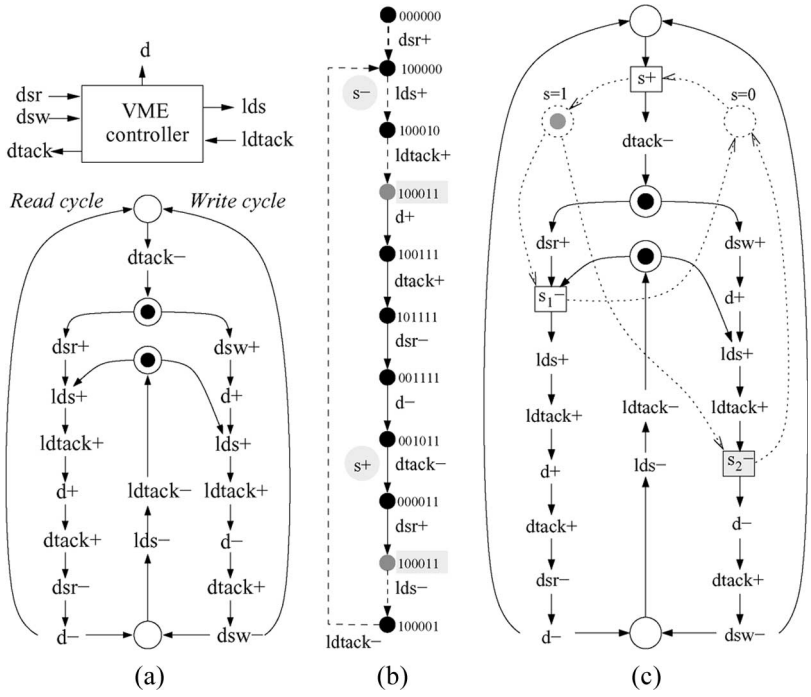


Fig. 2. (a) VME bus controller. (b) Trace connecting an encoding conflict. (c) Signal insertion.

A PN is reversible if its reachability graph is strongly connected. Systems that are not reversible often have some initialization sequences that lead to a cyclic behavior. For simplicity, we will assume that our PNs are reversible. The extension of the methods in this paper to nonreversible nets is straightforward and will be shortly discussed.

An STG is a triple $\langle N, \Sigma, \Lambda \rangle$, where N is a PN, Σ is a set of signals (input, output, and internal), and Λ is the labeling function Λ: $\mathcal{T} \rightarrow \Sigma \times \{+, -\}$. Transitions are interpreted as signal changes. a^+ and a^- respectively denote the rising and falling transitions of a signal $a \in \Sigma$, whereas a^* denotes a generic rising or falling transition.[1] An example of STG is shown in Fig. 2(a). In this STG, three entities (the bus, the device, and the controller) interact through a bidirectional buffer according to a given protocol. For simplicity, those places that only have one predecessor transition and one successor transition are not depicted. In that case, the tokens are held on the corresponding arcs.

B. State Encoding

Each marking of an STG is encoded with a binary vector of signal values through a labeling function $\lambda : [m_0\rangle \rightarrow \{0,1\}^{|\Sigma|}$. The notation $\lambda_s(m)$ will be used to denote the value of signal s at marking m. All markings must consistently be encoded by λ, i.e., no marking m can have an enabled rising (falling) transition $a^+(a^-)$ if $\lambda_a(m) = 1(\lambda_a(m) = 0)$.

An STG is said to satisfy the complete state coding (CSC) property if, when the same binary code is assigned to two different markings, the set of internal and output signals enabled at each marking is the same. The CSC property is a necessary condition for the correct implementation of an STG specification. When the CSC property holds, the events that the circuit must produce at each reachable state are uniquely determined by the binary code of the state. Encoding conflicts can be detected by using existing techniques based on PN unfoldings [18] or ILP models [2]. These methods would reveal the conflict, as shown in Fig. 2(b), between the states with the shadowed encoding 100011, corresponding to the vector (dsr, dsw, dtack, d, lds, ldtack). A trace that connects the conflicting states would also be reported by the aforementioned methods.

Four conditions are required for an STG to be implementable under the speed-independent delay model: consistency, CSC, output persistency, and boundedness [8].

Therefore, to implement the protocol represented by the STG in Fig. 2(a), the encoding problem must be solved. Notice that the underlying PN of the STG does not belong to any of the classes, for which there are structural methods for the encoding problem, i.e., marked graphs or free-choice nets. With the current state of the art, the encoding problem has to be solved with state-based methods that may suffer from the state explosion problem. The method proposed in this paper is the first to solve the encoding problem directly on the structure of the PN.

To solve the conflict, a new signal s can be inserted. Two possible insertion points for s^+ and s^- are depicted in Fig. 2(b). These points can easily be identified with their corresponding events in the STG (e.g., dtack$^-$ for s^+ and lds$^+$ for s^-). Fig. 2(c) depicts how events s^+ and s_1^- have been inserted before dtack$^-$ and lds$^+$, respectively (we ask the reader to ignore event s_2^+ and the dotted places and arcs for the moment). This insertion disambiguates the conflict depicted in Fig. 2(b).

[1]We will use subindices to denote different occurrences of the same signal transition, e.g., a_1^+, a_2^+.

In general, other conflicts can also be solved as a by-product of any signal insertion.

However, any signal must have a consistent encoding. Our example has a choice that selects between the read and write cycles of the VME bus, and the insertion of signal s has broken a conflict in the read cycle. Unfortunately, this insertion is inconsistent with the write cycle, since traces with two consecutive rising transitions of s are possible, e.g.,

$$\text{dsw}^+ d^+ \text{lds}^+ \text{ldtack}^+ d^- \text{dtack}^+ \text{dsw}^- \boxed{s^+} \text{dtack}^- \text{lds}^- \text{ldtack}^-$$

leading to the same initial state, in which the same trace can occur without having fired any s^- event.

How can s^+ and s^- events be inserted in such a way that, aside from solving the targeted encoding conflicts, they guarantee a consistent behavior for any feasible trace of the system?

Any expert designer would immediately detect that another s^- event is required and would manually insert the s_2^- event to guarantee consistency, as shown in Fig. 2(c). However, PNs can have very intricate causality, concurrence, and choice relations that make the behavioral analysis difficult, unless one enumerates the state space explicitly.

The main contribution of this paper is a method for automatically inserting all the events of a new signal in such a way that consistency is guaranteed. The method is based on the structural theory of PNs, which is presented as follows.

C. PNS and Linear Algebra

Given an occurrence sequence $m_0 \xrightarrow{\sigma} m$, the number of tokens for a place p in m is equal to the tokens of p in m_0 plus the tokens added by the input transitions of p in σ minus the tokens removed by the output transitions of p in σ. If we denote $\#(\sigma, t)$ as the number of times that a transition t occurs in σ, we can write the marking equation for p as follows:

$$m(p) = m_0(p) + \sum_{t \in {}^\bullet p} \#(\sigma, t)\mathcal{F}(t, p) - \sum_{t \in p^\bullet} \#(\sigma, t)\mathcal{F}(p, t).$$

The marking equations for all the places in the net can be written in the following matrix form [see Fig. 1(c)]:

$$m = m_0 + \mathbf{N} \cdot \vec{\sigma}$$

where $\vec{\sigma} = (\#(\sigma, t_1), \ldots, \#(\sigma, t_n))$ is called the Parikh vector of σ, and $\mathbf{N} \in \mathcal{Z}^{P \times T}$ is the incidence matrix of the net

$$\mathbf{N}(p, t) = \mathcal{F}(t, p) - \mathcal{F}(p, t).$$

If a marking m is reachable from m_0, then there exists a sequence σ such that $m_0 \xrightarrow{\sigma} m$, and the following system of equations has at least the solution $X = \vec{\sigma}$:

$$m = m_0 + \mathbf{N} \cdot X. \tag{1}$$

If (1) is infeasible, then m is not reachable from m_0. The inverse does not hold in general: there are markings that satisfy (1) but are not reachable. Those markings are said to be spurious [31]. Fig. 1(a)–(c) presents an example of a net with spurious markings: the Parikh vector $\vec{\sigma} = (2, 1, 0, 0, 1, 0)$ and the marking $m = (0, 0, 1, 1, 0)$ are a solution to the marking equation, as shown in Fig. 1(c). However, m is not reachable by any feasible sequence. Fig. 1(b) depicts the graph that contains the reachable markings and the spurious markings (shadowed). The numbers inside the states represent the tokens at each place $(p_1, \ldots, p_5)$. This graph is called the potential reachability graph. The initial marking is represented by the state (1, 0, 0, 0, 0). The marking (0, 0, 1, 1, 0) is only reachable from the initial state by visiting a negative marking through the sequence $t_1 t_2 t_5 t_1$, as shown in Fig. 1(b). Therefore, (1) provides only a sufficient condition for the reachability of a marking.

For certain subclasses of PNs, e.g., a free-choice [26], live, bounded, and reversible net, (1), together with a collection of sets of places (called trap invariants) of the system, completely characterizes reachability [9]. For the rest of the cases, the problem of the spurious solutions can be palliated by using trap invariants [11] or adding some special places, i.e., cutting implicit places [31], to the original PN that remove spurious solutions from the original marking equation. Unfortunately, in general, it is not possible to remove all spurious solutions from the marking equation [31]. See Section III-E for a further discussion on this limitation of our approach.

D. Implicit Places

The key ingredient of this paper is the use of implicit places for signal insertion. Intuitively, a place p is implicit if it is never the unique place that prevents the firing of a transition. A sufficient condition for a place $p \in P$ to be implicit is the nonexistence of a solution for m, σ, and s to the following LP model [31], where $P' = P \backslash \{p\}$:

$$\begin{aligned} m - \mathbf{N} \cdot \sigma &= m_0 \\ m[P'] - \mathcal{F}[P', p^\bullet] \cdot s &\geq \mathbb{0} \\ m(p) - \mathcal{F}[p, p'] \cdot s &< 0 \\ \mathbb{1} \cdot s &= 1 \\ m, \sigma, s &\geq \mathbb{0} \end{aligned}$$

The model is interpreted as follows. The vector s is a "transition selector," since the constraints $\mathbb{1} \cdot s = 1$ and $s \geq 0$ imply that s has exactly one element at 1. The first constraint corresponds to the marking equation and indicates that m is potentially reachable (a necessary condition). The second constraint indicates that all predecessor places, except p, of the transition that was selected by s have enough tokens to enable it.[2] The third constraint indicates that p does not have enough tokens to enable the transition, thus preventing it from firing. Therefore, if this model has a solution, the place might not be implicit. On the other hand, the inexistence of a solution guarantees implicitness. The fact that the condition is only sufficient comes from the fact that the marking m could be spurious.

[2]The notation $\mathcal{F}[P', p^\bullet]$ denotes a matrix covering all places (rows) in P' and all transitions (columns) in $p^\bullet$.

The previous LP model can slightly be transformed and converted to the dual problem (see the details in [31]). Finally, the following result can be derived.

Theorem 2.1 [31]: If $m_0(p)$ is greater than or equal to the optimal value of the following LP problem, then p is implicit:

$$\begin{aligned} \min\ & \mathbf{y}\cdot m_0[P'] + \mu \\ & \mathbf{y}\cdot \mathbf{N}[P',T] \le \mathbf{N}[p,T] \\ & \mathbf{y}\cdot \mathcal{F}[P',p^\bullet] + \mu\cdot \mathbb{1} \ge \mathcal{F}[p,p^\bullet] \\ & \mathbf{y} \ge \mathbb{0}\,. \end{aligned} \tag{2}$$

Going back to the example of the VME controller, we now give an intuitive idea of the consistent signal insertion technique presented in this paper. In a consistent STG, each signal can be modeled by a pair of encoding places p_0 and p_1 that indicate when the signal value is 0 (p_0 marked) or 1 (p_1 marked). In Fig. 2(c), they are called $\langle s=0\rangle$ and $\langle s=1\rangle$ for signal s and are represented as dotted places. These places only interact with the events of signal s with arcs $\langle s=0\rangle \to s^+ \to \langle s=1\rangle$ and $\langle s=1\rangle \to s^- \to \langle s=0\rangle$.

The reader can immediately realize that the behavior of the PN in Fig. 2(c) is the same, regardless of the presence of the dotted places. Therefore, the dotted places are implicit. The proposed method for signal insertion is based on a result in [13].

Theorem 2.2 [13]: Given an STG with signal s, two new places s_0 and s_1 are added such that every $s^+(s^-)$ transition puts a token into place $s_1(s_0)$ and removes a token from $s_0(s_1)$, and no other transitions are connected to them. Signal s is consistent if and only if places s_0 and s_1 are implicit.

This result is crucial for the method presented in this paper. The reader can realize that the dotted places would not be implicit if the dotted arcs $\langle s=1\rangle \to s_2^- \to \langle s=0\rangle$ would not be present in the STG with the dotted places.

The result from Theorems 2.1 and 2.2 enables us to not only check whether a signal is consistently encoded but also automatically find the insertion points that guarantee consistency. The problem can be solved by using ILP models based on the LP model from Theorem 2.1. Most of this paper will be devoted to formalizing the method for consistently automatically inserting state signals in an STG.

E. Concurrency and Lock Relations

Concurrency and lock relations will be important criteria for selecting insertion points for new signals. These relations will be incorporated in the cost function of the ILP model presented in Section IV.

Two transitions t_i, t_j are concurrent if there exists a reachable marking m such that $m[t_i t_j\rangle$ and $m[t_j t_i\rangle$. Although computing the concurrence relation in a safe PN is PSPACE-complete [5], there is an efficient algorithm that can approximate the concurrence relation [20] and is exact for the class of free-choice PNs.

A pair of signals a, b are in a lock relation if the firing of the events of a and b alternates, i.e., between the firing of two events of one of the signals, there is always a firing from the other signal. Lock relations have been used as sufficient conditions for CSC in some classes of PNs [35]. The existence of lock relations helps in avoiding CSC conflicts and reducing the logic for the implementation of output signals.

III. Solving CSC Conflicts

A. Example

Let us describe the approach with an example. The initial STG and its state graph are shown in Fig. 4(a) and (c). The conflicting states m_1 and m_2 are encoded with label 0000. The sequences that connect the states are $\sigma_1 = x^+a^+x_1^-a^-$ and $\sigma_2 = y_1^+y^-$. Events of signal s must be inserted in these sequences to disambiguate the encoding conflict. In principle, more than one s^+ and s^- events could be inserted at every sequence, but the number of s^+ events in σ_1 must be one more than the number of s^- events. Similarly, the number of s^- events in σ_2 must be one more than the number of s^+ events. This guarantees that $\lambda_s(m_1)=0$ and $\lambda_s(m_2)=1$.

Let us assume that we have decided to break the sequences σ_1 and σ_2 by the transitions a^- and y^-, respectively. The places associated with the new signal s will have arcs that are connected to these transitions, as shown by the dotted lines in Fig. 4(d). Accordingly, the chosen transitions are overloaded with the corresponding label: transitions a^- and y^- now become s^+; a^- and s^-; y^-, respectively. However, these new events are not sufficient for s to be consistent: the sequence $b^+x_2^-y_2^+b^-s^-y^-x^+$ can infinitely be repeated from the initial state, and only s^- events appear for signal s.

This inconsistency is related to the nonimplicitness of place $\langle s=0\rangle$. Notice that $\langle s=0\rangle$ is connected to the same transitions as the implicit place p in Fig. 4(b) but x_2^-. The consistency of s can be achieved by adding the arc $\langle s=0\rangle \to x_2^-$ and the complementary arc $x_2^- \to \langle s=1\rangle$. This insertion point must be labeled with an occurrence of s^+, i.e., transition x_2^- now becomes $s^+; x_2^-$.

Therefore, the new events are inserted, overloading the events of existing transitions. The overloaded transitions can afterward be expanded by well-known PN rules (series or parallel expansion) that preserve all the necessary properties for synthesis [26]. The final STG is shown in Fig. 4(e), where places $\langle s=0\rangle$ and $\langle s=1\rangle$ have been removed, because they are implicit. The final STG has CSC.

B. Main Algorithm

The main algorithm for solving CSC is presented in Fig. 3. Finding the conflicts and the corresponding traces can be done by using existing methods to check for CSC [2], [18]. The core of the algorithm is the creation of the implicit places s_0 and s_1 that determine the location of the events for the new state signal. This will be discussed in the next section.

After having found the insertion points for the new s^+ and s^- events, the transitions at these points are overloaded with these new labels. The transition t at every insertion point will also have a label a^* from another signal. In case a^* is an input event, it is not possible to insert s^* as a preceding event, since this would change the input/output interface of the specification. In

while CSC conflicts exist **do**
- Find conflict (m_1, m_2) and traces $m_1 \xrightarrow{\sigma_1} m_2 \xrightarrow{\sigma_2} m_1$
- Find implicit places, s_0 and s_1 that break the conflict and guarantee consistency of a new signal s
- **for every transition** t with arcs $s_0 \rightarrow t \rightarrow s_1$ **do** Relabel the transition as $(t; s^+)$ or $(s^+; t)$
- **for every transition** t with arcs $s_1 \rightarrow t \rightarrow s_0$ **do** Relabel the transition as $(t; s^-)$ or $(s^-; t)$
- Expand (series or parallel) the relabeled transitions

Fig. 3. Main algorithm for solving CSC conflicts.

that case, the new s^* event is inserted as a successor event.[3] In this paper, we only deal with the sequential insertion of events. Insertions with more concurrence are also possible and are applied if the goal is to improve performance and the transition that was selected belongs to the critical cycle of the system. The Appendix describes this enhancement of the method.

C. ILP Model to Find Insertion Points (Implicit Places)

The method for inserting a new state signal is based on the LP model (2) in Theorem 2.1. Instead of checking for implicitness, we aim at finding two implicit and complementary places that mimic the value of a new state signal. To achieve this, model (2) is transformed into a new ILP model as follows:

1) Two new sets of $|T|$ variables ($\mathcal{F}(T, s_0)$ and $\mathcal{F}(s_0, T)$) are added to the ILP model to encode $\mathbf{N}[s_0, T]$ as $\mathcal{F}(T, s_0) - \mathcal{F}(s_0, T)$, with $\mathbf{N}(s_0, t) \in \{-1, 0, 1\}$. For the sake if simplicity, in the rest of this paper, we will only refer to variables $\mathbf{N}[s_0, t]$. These variables represent the connections of the implicit place s_0 to the locations in the PN where the new events will be inserted. There is no need to add rows for s_1, since it is complementary to s_0, i.e., $\mathbf{N}[s_0, T] = -\mathbf{N}[s_1, T]$.
2) Using the conflict traces σ_1 and σ_2, the new constraints

$$\#(\sigma_1, \mathbf{N}[s_1, T]) = \#(\sigma_1, \mathbf{N}[s_0, T]) + 1$$
$$\#(\sigma_2, \mathbf{N}[s_0, T]) = \#(\sigma_2, \mathbf{N}[s_1, T]) + 1$$

where

$$\#(\sigma, \mathbf{X}) = \sum_{\mathbf{X}[t]>0} \#(\sigma, t)$$

guarantee that $\lambda_s(m_1) = 0$ and $\lambda_s(m_2) = 1$. They imply that σ_1 must contain one more occurrence of s^+ than s^-, and similarly, with σ_2 and the complementary events, $\mathbf{N}[s_1, T]$ can be substituted by $-\mathbf{N}[s_0, T]$.[4]

[3]The cost function in our ILP model guarantees that an insertion point between input events is never provided.

[4]The term $\#(\sigma, \mathbf{N}[s_0, T]) = \sum_{\mathbf{N}[s_0,t]>0} \#(\sigma, t)$ cannot be transformed to a linear form if the $N[s_0, t]$ entries are variables in the range $\{-1, 0, 1\}$. This is the reason for introducing the two rows of nonnegative variables $\mathcal{F}(T, s_0)$ and $\mathcal{F}(s_0, T)$ instead of only one row $\mathbf{N}[s_0, T]$ of unrestricted variables. The term can then be expressed in a linear form as $\#(\sigma, \mathbf{N}[s_0, T]) = \sum_t \mathcal{F}(t, s_0) \cdot \#(\sigma, t)$.

Formally, the ILP model for solving for the insertion of a new state signal is

ILP model for solving conflict $m_1 \xrightarrow{\sigma_1} m_2 \xrightarrow{\sigma_2} m_1$:

$$\min \ \mathbf{y}_0 \cdot m_0[P'] + \mathbf{y}_1 \cdot m_0[P'] + \mu$$
1) Constraints from (2) for s_0 implicit ($\mathbf{y}_0$ is $\mathbf{y}$)
2) Constraints from (2) for s_1 implicit ($\mathbf{y}_1$ is $\mathbf{y}$)
3) $\#(\sigma_1, \mathbf{N}[s_1, T]) = \#(\sigma_1, \mathbf{N}[s_0, T]) + 1$
4) $\#(\sigma_2, \mathbf{N}[s_0, T]) = \#(\sigma_2, \mathbf{N}[s_1, T]) + 1$

$$\mathbf{y}_0, \mathbf{y}_1 \geq 0, \quad \mathbf{N}[s_0, T], \mathbf{N}[s_1, T] \in \{0, 1, -1\}^{|T|}. \qquad (3)$$

If model (3) is feasible and the initial markings of places s_0 and s_1 satisfy the necessary conditions, the variables that encode $\mathbf{N}[s_0, T]$, and $\mathbf{N}[s_1, T]$ represent the necessary arcs to make s_0 and s_1 implicit.

Theorem 3.1: If model (3) is feasible and $m_0[s_0]$, $m_0[s_1]$ can be assigned such that $m_0[s_0] + m_0[s_1] \leq 1$, $m_0[s_0] \geq \mathbf{y}_0 \cdot m_0[P']$, and $m_0[s_1] \geq \mathbf{y}_1 \cdot m_0[P']$, then s_0 and s_1 are implicit places in the PN represented by $(\mathbf{N}, m_0)$.

Proof: Let $\mathbf{y}_0$, $\mathbf{y}_1$, $\mathbf{N}[s_0, T]$, $\mathbf{N}[s_1, T]$ be a solution of the ILP model (3). Let us focus on the implicitness of s_0 (the reasoning for s_1 is identical). If $\mathbf{y}_0$ is a solution of (3), it particularly satisfies constraint 1) of the model, and therefore, $\mathbf{y}_0$ is also a solution to LP (2) when $\mathbf{N}$ is used as the incidence matrix. Notice that the cost function that was minimized in (3) is $\mathbf{y}_0 \cdot m_0[P'] + \mathbf{y}_1 \cdot m_0[P'] + \mu$, whereas the cost function in LP (2) for $p = s_0$ minimizes only the term regarding $\mathbf{y}_0$. This implies that the minimization in (3) might not be optimal for LP (2) when $\mathbf{N}$ is used as the incidence matrix, i.e., $\mathbf{y}_0 \cdot m_0[P'] \geq k_{\min}$, where $k_{\min}$ is the optimal value of LP (2) when $\mathbf{N}$ is used as the incidence matrix. This, together with $m_0[s_0] \geq \mathbf{y}_0 \cdot m_0[P']$, implies that $m_0[s_0] \geq k_{\min}$, and therefore, applying Theorem 2.1 s_0 is implicit. ■

Note that conditions regarding $m_0[s_0]$ and $m_0[s_1]$ are needed in Theorem 3.1 because model (3) cannot compute their value. For the example in Fig. 4(a), the ILP solver finds the solution

$$\begin{array}{r} \\ \mathbf{N}[s_0, T] = \end{array} \begin{array}{cccccccccc} \overbrace{x^+} & \overbrace{a^+} & \overbrace{x_1^-} & \overbrace{a^-}^{\sigma_1} & \overbrace{y_1^+} & \overbrace{y^-}^{\sigma_2} & x_2^- & y_2^+ & b^+ & b^- \\ [0 & 0 & 0 & -1 & 0 & +1 & -1 & 0 & 0 & 0]. \end{array}$$

The transitions with $\mathbf{N}[s_0, T] = -1$ accommodate label s^+, whereas the ones with $\mathbf{N}[s_0, T] = +1$ accommodate label s^-. It is important to realize that transition x_2^-, which does not belong to σ_1 or σ_2, is also defined as an insertion point for s^+ to preserve the consistency of the signal. After the insertion of signal s and the expansion of the relabeled transitions, the STG in Fig. 4(e) is obtained.

Finally, the cost function used in the ILP model is an extension of the one in model (2)

$$\min (\mathbf{y}_0 + \mathbf{y}_1) \cdot m_0[P'] + \mu + c_1 (\mathbf{N}[s_0, T]) + c_2 (\mathbf{N}[s_0, T])$$

where, aside from the necessary minimization that was required for the implicitness of place s_0, two new subfunctions c_1 and c_2 are used to guide the ILP solver toward finding good solutions.

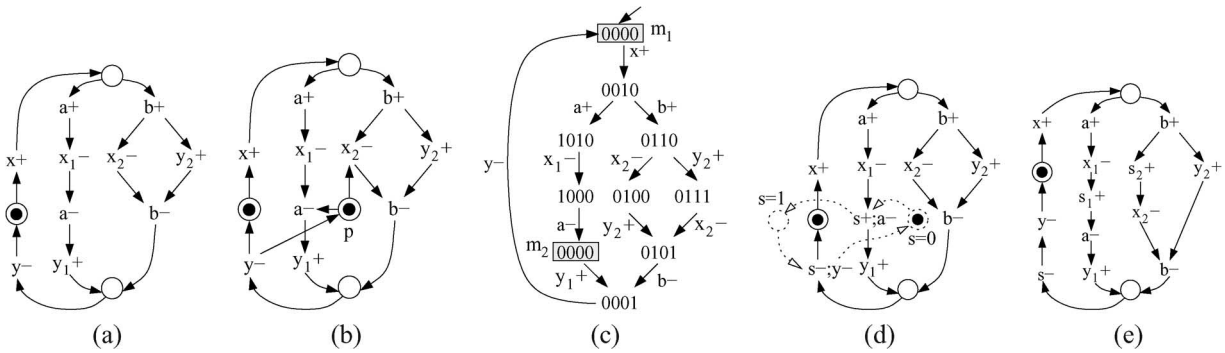


Fig. 4. (a) Initial STG. (b) Implicit place p. (c) CSC conflict. (d) Incorrect insertion of s. (e) Final STG.

Furthermore, area or performance goals can be considered in the cost function. Section IV contains a detailed description of the cost function.

The complexity of the method mainly corresponds to the difficulty of solving the ILP problems that correspond to each iteration from the algorithm in Fig. 3. The number of variables of each ILP problem is linear to the size of the PN: $2 \cdot |P|$ variables are needed to encode the implicitness of places s_0 and s_1 [from model (2)], and $2 \cdot |T|$ variables are needed to store $\mathbf{N}[s_0, T]$, as $\mathbf{N}(s_0, T) = \mathcal{F}(T, s_0) - \mathcal{F}(s_0, T)$. The number of constraints of each ILP problem is twice the number of the constraints needed for the LP model (2), which is $3 \cdot |T|$.

The method presented in this section can easily be extended to nonreversible systems by ignoring the trace σ_2 in the ILP model. This trace is not strictly necessary for the correct insertion of signals; however, it helps in finding better insertion points when the cost function is used for optimization.

D. Postprocess for Nonessential Signal Removal

As mentioned in the Introduction, there are very efficient methods for checking CSC in the literature (see [2] and [18]). Once CSC is achieved by the algorithm presented in Section III-B, one can use any of those methods to remove inserted signals that are not essential for having CSC. Nonessential signals can appear if there are conflicts that are disambiguated by two or more signal insertions. To check if a given signal is nonessential, it can simply be relabeled as silent, and the resulting STG can be checked for CSC. If CSC holds in the new STG, then the signal is not needed, and therefore, it can be removed from the net. Section V shows the significance of applying this simple greedy process, even for small benchmarks.

The insertion of nonessential signals can be avoided (or alleviated) if all the encoding conflicts with their corresponding conflict traces are computed first, and then, only the minimal set of conflicts that ensure correct encoding are solved. The ILP methods in [2], which were used to check the state encoding, work as follows. The system is free from encoding conflicts if a given model is infeasible. If the model is feasible, a solution that represents the conflict trace will be returned. Unless solved, this solution will always be the one that the ILP solver will compute. Hence, obtaining a different conflict trace requires changing the model, but modifying the model without affecting the feasibility or completeness is not an easy task. This prevents the methods in [2] from being easily adopted to compute all the conflicts at once.

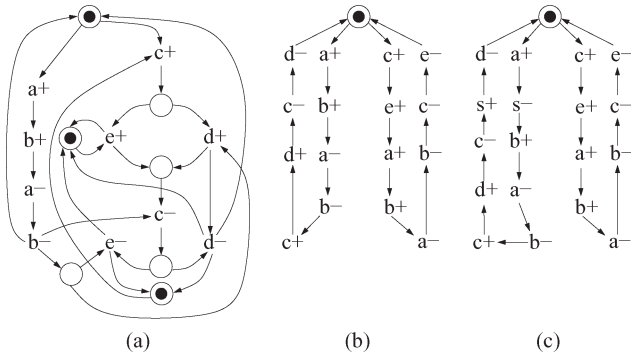


Fig. 5. Example with an unsolvable CSC conflict.

E. Incompleteness of the Method

The method presented in this paper cannot guarantee a solution, even in the case that it exists. The incompleteness of the method comes from the limitations of the structural theory for characterizing the reachable markings of a PN.

An example of this situation is depicted in Fig. 5 (a and c are inputs, whereas b, d, and e are outputs). The STGs in Fig. 5(a) and (b) have the same behavior. However, the one in Fig. 5(a) is more compact, since the transitions $a^+b^+a^-b^-$ represent two different subsequences of the behavior. A state-based tool such as **petrify** [8] can easily find a solution for Fig. 5(a), like the one in Fig. 5(c), with the insertion of the signal s. However, this solution requires restructuring the net (see Fig. 5(a)) by unfolding the shared subsequences and enabling a signal insertion in one of the branches. The method presented in this paper does not allow any restructuring of the net, and for this reason, it can find an appropriate signal insertion for Fig. 5(b) but not for Fig. 5(a).

Characterizing the class of STGs for which a solution can be guaranteed is a difficult problem that has the same nature as characterizing the reachable markings of a PN. However, we have a strong belief that well-structured nets with simple causality and concurrence relations are more amenable to

accept the small structural transformations that are required for solving the state encoding problem.

IV. Optimization for Area and Performance

In this section, we show how the CSC resolution technique presented in Section III can be guided to take area or performance into account.

A. Cost Function

The cost function of the model is designed to choose good insertion points. This has a strong impact on the quality of the solutions. By defining coefficients for the $\mathbf{N}[s_0, T]$ variables in the cost function, the obtained solutions are biased toward optimizing the following aspects:

1) Minimize the number of events of signal s.
2) Maximize the number of CSC conflicts that were solved in σ_1 and σ_2 (other conflicts, aside from the ones with m_1 and m_2, may exist).
3) Increase the number of lock relations with the other signals in σ_1 and σ_2 (lock relations are strongly related to the implementability of an STG as a speed-independent circuit, as shown in the literature [21], [34]).
4) If optimizing the area, minimize the number of concurrent transitions in the net.
5) If optimizing the performance, minimize the number of critical transitions.

The main idea behind maximizing the solved CSC conflicts is to reduce the number of extra signals that are needed to achieve CSC. By creating lock relations with the newly inserted signals, the logic usually becomes simpler.

Hence, apart from the minimization of the cost function from the LP model in Theorem 2.1, the cost function is also defined for each variable $\mathbf{N}[s_0, t]$ to minimize two subcosts functions c_1, c_2 for conflict sequences σ_1 and σ_2, respectively. In case of conflict $m_1 \xrightarrow{\sigma_1} m_2$, the cost function $c_1(\mathbf{N}[s_0, t])$ can be expressed as follows:

$$c_1\left(\mathbf{N}[s_0, t]\right) = k(t, \sigma_1) - k_{\text{csc}} \cdot \text{ncsc}(t, \sigma_1) - k_{\text{lck}} \cdot \text{nlck}(t, \sigma_1) \tag{4}$$

where $k(t, \sigma_1)$ is a positive integer K. If transition t is at the beginning or end of the sequence σ_1, or when $|({}^\bullet t)^\bullet| \geq 2$, we impose $k(t, \sigma_1) > K$. This prevents the selection of transitions that are in the borders of the conflict sequences or in a structural conflict, respectively. Selecting transitions that are in the borders of the conflict sequences may not break the conflict and is therefore discouraged. The other two terms of (4) represent the two general goals on the list above: $\text{ncsc}(t, \sigma_1)$ indicates the number of CSC conflicts that are broken in sequence σ_1 when inserting at the insertion point that is represented by t, and $\text{nlck}(a_i*, \sigma_1)$ represents the number of locked signals with signal a in the loop $\sigma_1\sigma_2$.[5] Factors k_{csc} and k_{lck} allow one to weight the importance of these two aspects in the cost function. In the experiments, we used $k_{\text{csc}} = 0.5$ and $k_{\text{lck}} = 0.2$. A symmetrical cost $c_2(\mathbf{N}[s_0, t])$ will be obtained for the conflict $m_2 \xrightarrow{\sigma_2} m_1$.

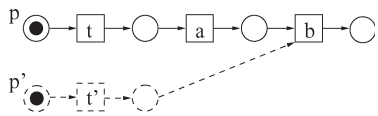
[5]Functions ncsc and nlck are normalized, and together with the fact that k_{csc} and k_{lck} are positive real numbers satisfying $k_{\text{csc}} + k_{\text{lck}} \leq 1$, this allows us to establish a value for K that is large enough to avoid degradated solutions of the model that can arise if $K < k_{\text{csc}} \cdot \text{ncsc}(t, \sigma_1) + k_{\text{lck}} \cdot \text{nlck}(t, \sigma_1)$.

B. Area and Performance Aware Insertion of Encoding Signals

Area: Given a specification, the complexity of its implementation strongly depends on the degree of concurrence that was observed: among all possible implementations, those with more concurrence also typically require more area. The reason is that concurrent systems have larger spaces of reachable states and, therefore, smaller don't-care sets for logic minimization. In the approach presented in this paper, if the goal is to optimize the area, signals are inserted as sequentially as possible. To accomplish this goal, the concurrence relations (see Section II-E) are used. The area-aware subcost functions c_i^{A} are computed from the ones in the previous section plus a new factor: $c_1^{\text{A}}(\mathbf{N}[s_0, t]) = c_1(\mathbf{N}[s_0, t]) + k_{\text{conc}} \cdot \text{nconc}(t)$, where $\text{nconc}(t)$ indicates how many transitions are concurrent with t.

Performance: Assuming that the information about which transitions are critical is available, i.e., for each transition t, the assertion $\text{critical}(t)$ indicates whether t is critical or not. A method for estimating this information, which is based on event simulation, is presented in the Appendix. To avoid increasing the cycle time (CT), a performance-aware subcost function c_i^{P} will be used. Function c_i^{P} is computed from c_i plus a new factor: $c_1^{\text{P}}(\mathbf{N}[s_0, t]) = c_1(\mathbf{N}[s_0, t]) + k_{\text{cr}} \cdot \text{critical}(t)$.

Moreover, when a solution that was found by solving the ILP model considers a critical transition t, the insertion can be performed by expanding the transition in parallel, instead of sequentially [26]. Once t has been expanded in parallel, the newly created transition (t' in the example in the following figure) is inserted as concurrent with the transitions in the conflict sequence as possible:



In the example, it is assumed that transitions t, a, and b belong to the conflict sequence and are noninputs. Instead of inserting t' as a predecessor of a, it is inserted as a predecessor of b, allowing t' to be concurrent not only to t but also to a. The method can be extended to consider other zones of the conflict sequence for concurrent transition insertions.

In general, the parallel insertion will not increase the CT, unless the transition that was inserted has a delay that is greater than t.

Area and performance are not independent factors, i.e., sometimes, area reduction implies reducing the logic within the critical paths, thus improving the latency of the system. The opposite can also occur: increasing the area can alleviate the length of the critical paths. This usually happens when a more concurrent circuit is obtained. The main objective of this section is to illustrate how the presented method can easily incorporate those optimization goals. The theory can be seen as a starting approach for more elaborated optimizations.

TABLE I
RESULTS (LITERALS/STATE SIGNALS)

example	petrify	ILP	ILP+
adfast	14/2	19/2	19/2
ircv-bm	38/2	43/3	43/3
mmu	29/3	29/4	26/3
mmu0	29/3	28/4	27/3
mmu1	32/3	23/2	23/2
mr0	45/3	37/4	30/3
mr1	35/4	29/3	29/3
nak-pa	18/1	18/1	18/1
nowick	14/1	14/1	14/1
par4	32/4	32/4	32/4
seq8	47/4	37/4	37/4
tsend-bm	39/2	44/3	44/3
alloc-outbound	16/2	16/2	16/2
duplicator	19/2	13/2	13/2
mod4_counter	25/2	28/3	28/3
ram-read-sbuf	18/1	18/1	18/1
sbuf-ram-write	22/2	33/3	32/2
sbuf-read-ctl	15/1	15/2	15/1
master_1882	38/1	38/1	38/1
trcv-bm	36/2	44/3	44/3
seq_mix	20/3	20/2	20/2
spec_seq4	20/3	19/2	19/2
total	601/52	597/56	585/51

V. EXPERIMENTAL RESULTS

The goal of the experimental results presented in this section is threefold: 1) evaluate the quality of the method by comparing the results with a state-based method; 2) evaluate the capability of handling large specifications, and 3) show the potential benefits of logic synthesis methods with regard to SDT.

Comparison With a State-Based Method: Table I evaluates the quality of the results for area optimization when compared to **petrify** [8]. The examples in [7] have been used.[6] The number of literals of the Boolean equations (in factored form) and the number of state signals that were inserted to solve CSC are reported.[7] Columns ILP+ and ILP report the results with or without the postprocess for eliminating nonessential signals described in Section III-D. In general, the levels of complexity of the circuits are similar, although the ILP+ approach has a moderate 6% improvement in literals. The new method (ILP) inserts more state signals than **petrify**. The reason is that **petrify** attempts to maximize the number of conflicts that were solved for each signal, whereas the new approach tries to reduce the number of events of each new signal. Overall, increasing the number of signals without increasing the number of literals is beneficial, since it provides a better initial decomposition that will reduce the number of additional signal decompositions for technology mapping, thus resulting in better mapped circuits. In other words, the average number of literals per signal decreases when the number of signals increases and the number of literals remains the same.

[6]Only four examples with irreducible conflicts have been omitted, since the conflicts cannot be solved, unless timing assumptions on the environment are used.

[7]Given that the examples are small, we do not report the CPU times, since they are negligible and irrelevant for the comparison.

Some of the extra signals that were inserted in the new approach can be regarded as nonessential (i.e., not needed for having CSC) by the greedy process shown in Section III-D and were consequently removed. The results of this postremoval of nonessential signals are shown in column ILP+. The overall complexity of the circuits was never increased when a nonessential signal removal was applied for the benchmarks used. On the contrary, in some examples (e.g., mr0), this greedy process significantly decreased the literal count.

Large Specifications: State-based methods cannot handle large STGs like the ones in Table II. This table shows the results for the two possible optimization goals described in this paper: area and delay. For the case of delay, the CT has been evaluated with the performance evaluation technique presented in the Appendix. For each of the optimization goals, results with or without the use of the corresponding heuristic (described in Section IV-B) are presented.

The benchmarks ART(n, m) model n pipelines of length m that were synchronized only on their starting points, thus exhibiting a high degree of concurrence. Benchmarks PPWK and PPARB are another type of pipelines, which were described in [18]. VAR(n, m) models the handshakes of a set of n read and m write processes into a 1-bit variable.

The rest of the examples are typical netlists of handshake components from an HDL like Tangram [1] or Balsa [10]. Those examples were obtained by hiding all the internal channels of the netlists and keeping only the events of the external signals:

1) $\mathbf{Par(12)}$: a 12-way parallelizer, as shown in Fig. 6.
2) $\mathbf{Seq(12)}$: a 12-way sequencer, with a tree structure that is identical to $\mathbf{Par(12)}$.
3) $\mathbf{SeqPar(21, 10)}$: a combination of 21 sequencers and ten parallelizers, as shown in Fig. 7.
4) $\mathbf{SPM(7, 16, 18)}$: a combination of sequencers, parallelizers, and mixers, as shown in Fig. 8. The actual circuit has an eight-way sequencer on the top (implemented as a tree of seven two-way sequencers), eight three-way parallelizers in the middle (implemented with 16 two-way parallelizers), and six four-way mixers (implemented with 18 two-way mixers).

The columns report the number of places, transitions, and input/output signals of the STG, respectively. The CPU time for encoding when applying the algorithm presented in this paper is also reported. For synthesis, structural methods that are based on projections [2] have been used. None of the previously existing techniques has been able to solve the encoding problem with examples of such size. For state encoding, the greedy postprocess for nonessential signal removal, which is presented in Section III-D, has been applied. It is remarkable to realize that the tool has been able to solve one of the examples [ART(20,9)] by inserting 39 signals.

Based on Table II, it can be observed that for most of the examples, there is a tradeoff between the area and the delay optimization: gains in performance usually require an increase in area. There are some examples [ART(10,9), ART(20,9), and SEQ(12)], however, where this principle does not hold. In these cases, the increase of concurrence implies a significant area

TABLE II
STATE ENCODING OF LARGE CONTROLLERS. AREA AND DELAY OPTIMIZATIONS

			Area Optimization					Delay Optimization				
			Without $nconc\ (t)$		With $nconc\ (t)$			Without $critical\ (t)$		With $critical\ (t)$		
benchmark	P=T	in/out	lit/sig	CT	lit/sig	CT	CPU	lit/sig	CT	lit/sig	CT	CPU
ART(10,9)	216/198	0/99	305/22	44.5	264/19	35.5	9'33"	481/24	45.0	492/24	41.4	1h36'
ART(20,9)	436/398	0/199	562/39	40.1	544/39	35.6	1h04'	768/46	44.0	852/56	41.6	2h05'
PPWK(3,12)	142/74	0/37	183/2	23.2	183/2	23.2	13"	183/2	23.2	191/2	20.2	13"
PPARB(3,12)	164/90	3/40	206/2	53.4	206/2	53.4	9"	206/2	53.4	210/2	45.9	24"
VAR(9,5)	302/338	73/77	319/19	563.6	319/19	563.6	14'58"	337/19	569.7	319/19	563.6	15'40"
VAR(12,1)	368/394	86/97	391/25	675.3	391/25	675.3	35'27"	391/25	675.3	391/25	675.3	36'07"
PAR(12)	63/52	13/13	119/13	85.8	101/12	72.9	6"	146/12	46.8	116/12	40.5	13"
SEQ(12)	52/52	13/13	73/10	224.0	59/6	202.4	6"	109/10	218.4	89/7	205.4	8"
SEQPAR(21,10)	160/128	32/32	263/19	157.5	261/19	156.1	2' 40"	281/19	145.0	291/19	150.8	4'48"
SPM(7,16,18)	192/144	30/30	242/17	222.9	229/14	207.9	1'55"	253/14	203.4	232/14	202.9	2'40"

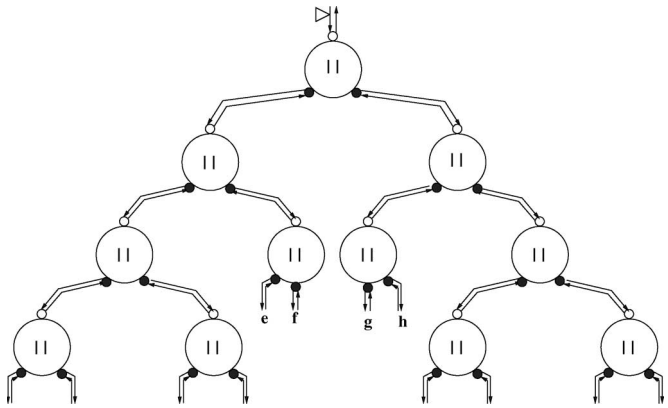


Fig. 6. Parallelizer $\mathbf{Par(12)}$.

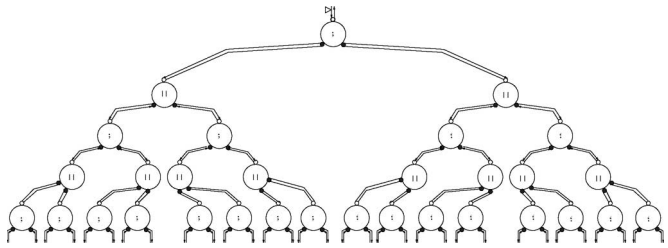


Fig. 7. Sequencers and parallelizers $\mathbf{SeqPar(21,10)}$.

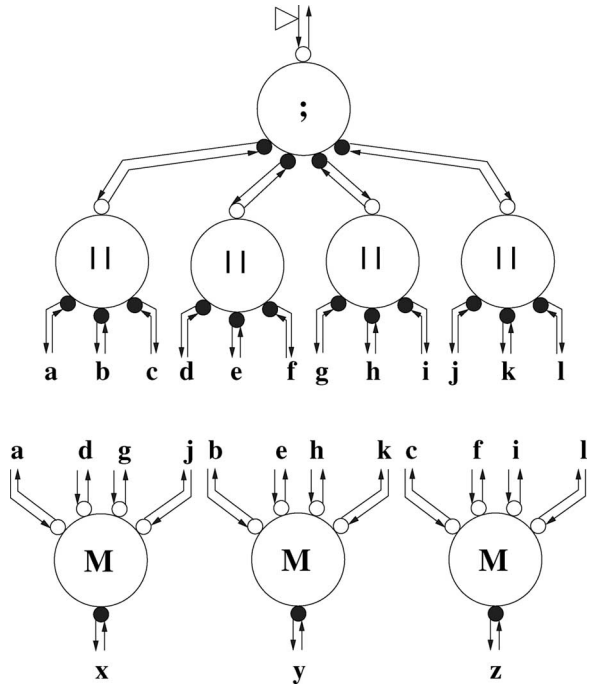


Fig. 8. Sequencers, parallelizers and mixers $\mathbf{SPM}$.

overhead that has a negative impact on the logic delays. The parallelism at the level of events does not compensate the complexity of the logic, thus making the circuits slower.

Heuristics: Table II also shows the impact of using the heuristics for optimizing the area and performance presented in Section IV-B. The two experiments executed for each benchmark aim at optimizing either the area or the performance. For the area optimization, the significance of using the concurrence relations when selecting insertion points for state encoding is shown: the option taking this factor (column With nconc(t)) into account has less literals than not considering the concurrence relations (column Without nconc(t)). A similar circumstance happens for the delay optimization: column Without critical(t) shows the delay optimization without considering information on critical events, inserting events in parallel if possible. In general, the information of criticality helps in alleviating the degradation of the CT of the system.

In general, by customizing the weighted terms in the cost function, one could potentially find solutions that trade off both characteristics.

There is a significant CPU overhead in the ART examples, especially in ART(20,9). This shows that the effectiveness of the ILP approach can be sensitive to the net structure and the size of the problem. The experiments were performed with a commercial solver [16].

Logic Synthesis Versus SDT: The major question is: how much is left for optimization by using logic synthesis for specifications generated by SDT from HDLs? Here, we will demonstrate that there is significant room for optimization.

We will use an n-way parallelizer as an example. Fig. 9(a) depicts a netlist of handshake components for a four-way parallelizer. Each channel c is implemented by two handshake signals: c_0 and c_1. In the conventional implementation of a parallelizer, two internal signals are required for encoding the state of the component. Each one is associated with one of the output channels. Fig. 9(c) shows the behavior of the netlist,

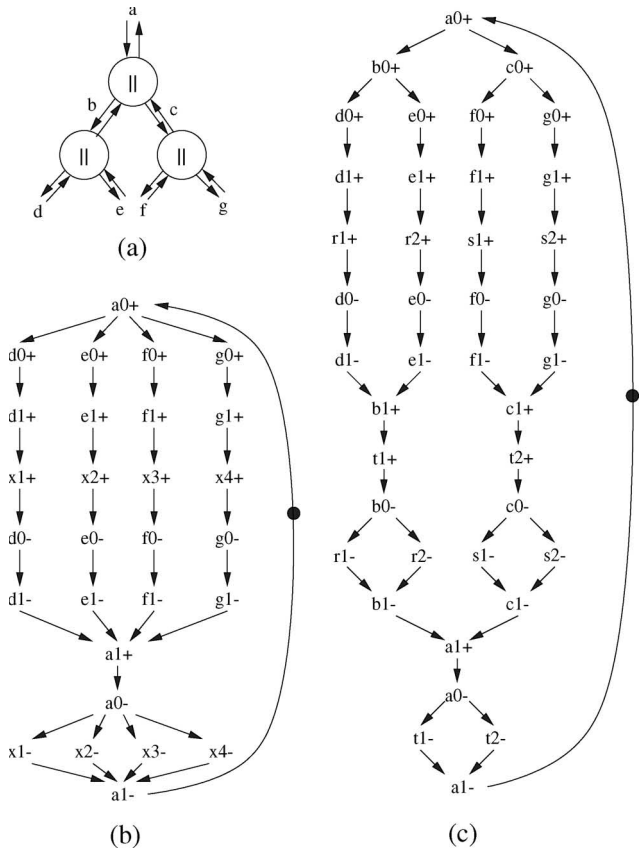


Fig. 9. Implementation of a four-way parallelizer. (a) Handshake components. (b) STG from logic synthesis. (c) STG from SDT.

TABLE III
CHARACTERISTICS OF AN n-WAY PARALLELIZER IMPLEMENTED USING SDT AND LOGIC SYNTHESIS

	Non-input signals	Critical cycle (all events)
SDT	$5(n-1)$	$6\lceil \log_2 n \rceil + 4$
Logic synthesis	$3n+2$	10

including the transitions of the internal channels and signals. Fig. 9(b) describes the behavior of the system after hiding the internal channels and adding state signals. This specification is the one obtained by logic synthesis tools after state encoding.

Therefore, Fig. 9(b) and (c) represents the behavior of the circuits obtained by logic synthesis and SDT, respectively. Table III describes some characteristics of a general n-way parallelizer: the number of noninput signals of the circuit and the number of critical events.[8] The following conclusions can be drawn:

1) The number of noninput signals is linear to n in both cases; however, the complexity is significantly larger for the syntax-directed approach ($5n$ versus $3n$).
2) The length of the critical cycle is constant in logic synthesis; however, it logarithmically increases in the syntax-directed approach in the case of a well-balanced tree. This growth is directly related to the number of levels in the tree of handshake components.

[8]See the Appendix for the definition of critical events.

We now illustrate the effectiveness of logic synthesis in different netlists of handshake components. Table IV compares four possible synthesis approaches: column "HDL" reports literals for a Tangram implementation,[9] whereas column "SDT" contains the results for the synthesis of the STGs that were generated from the corresponding HDL specifications with all the internal channels. Columns "ILP Area Optimization" and "ILP Delay Optimization" present the results for the same specifications but where internal channels are hidden and the encoding method of this paper is applied afterward, optimizing for the area and performance, respectively. The CT is also reported for each approach. Some conclusions can be drawn:

1) There is a tangible reduction in the complexity of the circuits. This reduction is mainly produced by the reencoding done by logic synthesis tools, i.e., the internal channels and state signals are eliminated, and new state signals are added for synthesis. The complexity of the circuits has been estimated by the number of literals of the Boolean equations. The reduction in area also has a similar impact in power consumption, since the number of events is reduced. Moreover, the power of logic synthesis can be observed if column HDL is compared to the rest of approaches. The results show drastic improvements, e.g., up to 63% in the number of literals for SPM(7,16,18) with respect to the ILP approaches. Even when only logic synthesis is applied without reencoding (column SDT), the simplification of the equations is significant with respect to HDL.
2) The most important result is with regard to performance. Handshake circuits are overencoded, since they are designed for a proper behavior under any possible environment. However, when embedded in a particular environment, significant reductions in complexity and delay can be achieved. The number of events in the critical cycle of the system is drastically reduced (almost by a factor of 3). This has a direct impact on the CT of the circuit.

From the above results, one can conclude that there is enough room for optimization by using the techniques presented in this paper, especially with regard to performance. These techniques can be combined with other approaches for optimization that work at higher levels of abstraction, like the clustering methods presented in [4].

VI. CONCLUSION

By solving the state encoding problem, logic synthesis can be incorporated into the main design flow of large asynchronous controllers. The method presented in this paper is a crucial step toward taking advantage of the optimizations that logic synthesis can do in the Boolean domain. This approach opens new opportunities to integrate logic synthesis with high-level design frameworks that generate asynchronous controllers by SDT.

[9]The cost of sequencers, parallelizers, and mixers is 8, 21, and 12, respectively [1]. The cost of a C element is considered to be five literals ($c^+ = ab + c(a+b)$).

TABLE IV
SYNTHESIS RESULTS FOR DIFFERENT NETLISTS OF HANDSHAKE COMPONENTS

	HDL		SDT		ILP			
					Area Opt.		Delay Opt.	
	lit	CT	lit	CT	lit	CT	lit	CT
PAR(12)	231	81.1	188	77.1	101	72.9	116	40.5
SEQ(12)	88	287.0	88	270.8	59	202.4	89	205.4
SEQPAR(21,10)	378	288.2	340	272.9	261	156.1	291	150.8
SPM(7,16,18)	608	583.4	431	579.4	229	207.9	232	202.9

APPENDIX
PERFORMANCE EVALUATION

We present the strategies used in this paper to estimate the critical events and the performance of a concurrent system. The strategies are related to previous work on determining the average time separation of events in asynchronous systems [24], [37].

A. Estimation of Critical Events

For state encoding that aims at performance optimization, it is necessary to detect the critical events of the specification. A simple strategy for deriving the critical skeleton of a concurrent system is next presented.

The strategy is based on a simulation of the system by using estimated delays for the transitions and estimated choice probabilities for the choices. It is possible to use probability distribution functions for any of them.

By performing an event-driven simulation of the system, the following information can be obtained for every transition t and every predecessor transition t':

$$\texttt{trig}(t', t) = \frac{\text{Number of times } t' \text{ triggered } t}{\text{Number of firing times of } t}.$$

We say that a transition t' triggers t when t' is the last to arrive before the enabling of t. Based on this information, it is possible to detect those events that will mostly likely be critical by defining a triggering threshold $\varepsilon \in [0, 1]$. For example, $\varepsilon = 0.1$ will select those events that trigger another event more than 10% of the times.

The algorithm for selecting the critical events works as follows.

1) Perform an event-driven simulation to calculate $\texttt{trig}(t', t)$ for all pairs of adjacent transitions.
2) Select $\text{Tr} = \{t' | \exists t : \texttt{trig}(t', t) \geq \varepsilon\}$. Remove all the transitions that are not in Tr from the PN.
3) Remove all the remaining transitions that are not in cycles.

The resulting PN will be a strongly connected subnet of the original PN that contains the skeleton of critical events.

Fig. 10 illustrates this calculation. For simplicity, all events are assumed to have a unit delay. The topmost place is a choice with three branches and probabilities 0.35, 0.6, and 0.05, respectively. The triggering threshold ε is 0.1.

After the simulation, the shadowed transitions are the ones in Tr. For example, i never triggers m, since j is always the last predecessor of m to arrive. On the other hand, l is a trigger of n but only for 5% of the times. Given that $\varepsilon = 0.1$, transition l is considered to be noncritical.

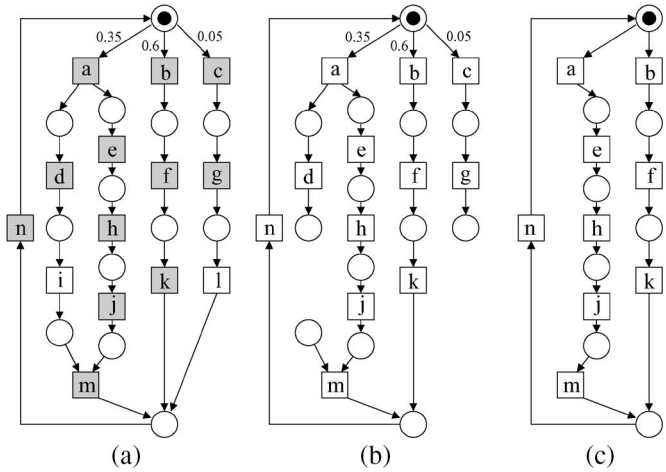


Fig. 10. (a) Triggering events (shadowed boxes). (b) After the removal of nontriggering events. (c) Critical events.

Fig. 10(b) depicts the remaining PN after the removal of the events that are not in Tr. Finally, Fig. 10(c) shows the final PN after removing the events that are not in cycles.

B. Estimation of Event Delays

The detection of critical events requires an estimation of the delay of each event of the system. The accuracy of the estimation highly depends on the available information about the implementation of each event. The technology-independent delay estimation used in this paper is the following:

$$\delta = \delta_{\text{inv}}(1 + \log_2 n)$$

where δ_{inv} is the estimated delay of an inverter, and n is some estimation of logic complexity. For example, if n is the number of literals of a cube, then $\log_2 n$ is the number of levels of an implementation with two-input gates.

Several scenarios can be distinguished for delay estimation.

1) Before solving CSC, when the circuit cannot be implemented. A good measure is the number of trigger signals that has a high correlation with the logic required for implementing a signal. Thus, the delay of each event is estimated by the previous formula, with n being the number of trigger events.
2) After solving CSC and before technology mapping. A better estimation for the delay of the events of a signal

can be obtained with the same formula, with n being the number of literals of the Boolean equation for that signal in factored form.

The delay for the input events depends on the behavior of the environment. The designer must provide an estimation based on the knowledge of the system. In the experiments, input events are considered slow, with a delay representing a couple of complex gates.

C. Performance Evaluation

Once the delay of each event has been calculated, the performance of a system can be calculated by analytical methods or simulations. Analytical methods are usually restricted to some subclasses of concurrent systems, e.g., marked graphs or free-choice nets.

Given a delay for each event, the performance of a system can simply be estimated by simulation. In our experiments, the CT is estimated as the average time separation between two consecutive firings of the most frequent event. Thus, if T is the simulation time and $\#e$ is the firing count of the most frequent event, the CT C is estimated as

$$C = \frac{T}{\#e}.$$

ACKNOWLEDGMENT

The authors would like to thank J. M. Colom for his suggestions and helpful discussions.

REFERENCES

[1] K. v. Berkel, *Handshake Circuits: An Asynchronous Architecture for VLSI Programming*, ser. International Series on Parallel Computation, vol. 5. Cambridge, U.K.: Cambridge Univ. Press, 1993.

[2] J. Carmona, J. M. Colom, J. Cortadella, and F. García-Vallés, "Synthesis of asynchronous controllers using integer linear programming," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 25, no. 9, pp. 1637–1651, Sep. 2006.

[3] J. Carmona and J. Cortadella, "State encoding of large asynchronous controllers," in *Proc. ACM/IEEE Des. Autom. Conf.*, Jul. 2006, pp. 939–944.

[4] T. Chelcea, A. Bardsley, D. Edwards, and S. M. Nowick, "A burst-mode oriented back-end for the Balsa synthesis system," in *Proc. DATE*, Mar. 2002, pp. 330–337.

[5] A. Cheng, J. Esparza, and J. Palsberg, "Complexity results for 1-safe nets," *Theor. Comp. Sci.*, vol. 147, no. 1/2, pp. 117–136, Aug. 1995.

[6] T.-A. Chu, C. K. C. Leung, and T. S. Wanuga, "A design methodology for concurrent VLSI systems," in *Proc. ICCD*, 1985, pp. 407–410.

[7] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev, "A region-based theory for state assignment in speed-independent circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 16, no. 8, pp. 793–812, Aug. 1997.

[8] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev, *Logic Synthesis of Asynchronous Controllers and Interfaces*. New York: Springer-Verlag, 2002.

[9] J. Desel and J. Esparza, *Reachability in Cyclic Extended Free-Choice Systems*, vol. TCS 114. Amsterdam, The Netherlands: Elsevier, 1993.

[10] D. Edwards and A. Bardsley, "Balsa: An asynchronous hardware synthesis language," *Comput. J.*, vol. 45, no. 1, pp. 12–18, 2002.

[11] J. Esparza and S. Melzer, "Verification of safety properties using integer programming: Beyond the state equation," *Form. Methods Syst. Des.*, vol. 16, no. 2, pp. 159–189, Mar. 2000.

[12] R. M. Fuhrer, B. Lin, and S. M. Nowick, "Symbolic hazard-free minimization and encoding of asynchronous finite state machines," in *Proc. ICCAD*, 1995, pp. 604–611.

[13] F. García-Vallés and J. M. Colom, "Structural analysis of signal transition graphs," in *Proc. Workshop PNSE—Modelling, Verification Validation*, D. H. I. B. Farwer and M. Stehr, Eds, 1997, pp. 123–134.

[14] J. Gu and R. Puri, "Asynchronous circuit synthesis with Boolean satisfiability," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 14, no. 8, pp. 961–973, Aug. 1995.

[15] D. A. Huffman, "The synthesis of sequential switching circuits," in *Sequential Machines: Selected Papers*, E. F. Moore, Ed. Reading, MA: Addison-Wesley, 1964.

[16] ILOG Inc. Solver CPLEX. [Online]. Available: http://www.ilog.com

[17] V. Khomenko, "Efficient automatic resolution of encoding conflicts using STG unfoldings," School Comput. Sci., Newcastle Univ., Newcastle, U.K., Report CS-TR-995, Jan. 2007.

[18] V. Khomenko, M. Koutny, and A. Yakovlev, "Detecting state coding conflicts in STG unfoldings using SAT," in *Proc. Int. Conf. Appl. Concurrency to Syst. Des.*, Jun. 2003, pp. 51–60.

[19] M. Kishinevsky, A. Kondratyev, A. Taubin, and V. Varshavsky, *Concurrent hardware: The theory and practice of self-timed design*, Series in Parallel Computing. Hoboken, NJ: Wiley, 1994.

[20] A. Kovalyov and J. Esparza, "A polynomial algorithm to compute the concurrence relation of free-choice signal transition graphs," in *Proc. Int. WODES*, 1996, pp. 1–6.

[21] K.-J. Lin, C.-W. Kuo, and C.-S. Lin, "Synthesis of hazard-free asynchronous circuits based on characteristic graph," *IEEE Trans. Comput.*, vol. 46, no. 11, pp. 1246–1263, Nov. 1997.

[22] K.-J. Lin, J.-W. Kuo, and C.-S. Lin, "Direct synthesis of hazard-free asynchronous circuits from STGs based on lock relation and MG-decomposition approach," in *Proc. Eur. Des. Test Conf.*, 1994, pp. 178–183.

[23] G. K. Maki and J. H. Tracey, "A state assignment procedure for asynchronous sequential circuits," *IEEE Trans. Comput.*, vol. C-20, no. 6, pp. 666–668, Jun. 1971.

[24] E. G. Mercer and C. J. Myers, "Stochastic cycle period analysis in timed circuits," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2000, vol. 2, pp. 172–175.

[25] G. D. Micheli, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Optimal state assignment for finite state machines," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. CAD-4, no. 3, pp. 269–285, Jul. 1985.

[26] T. Murata, "Petri nets: Properties, analysis and applications," *Proc. IEEE*, vol. 77, no. 4, pp. 541–574, Apr. 1989.

[27] S. M. Nowick and D. L. Dill, "Automatic synthesis of locally-clocked asynchronous state machines," in *Proc. ICCAD*, Nov. 1991, pp. 318–321.

[28] E. Pastor and J. Cortadella, "An efficient unique state coding algorithm for signal transition graphs," in *Proc. ICCD*, Oct. 1993, pp. 174–177.

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

[30] M. Schäfer and W. Vogler, "Component refinement and CSC solving for STG decomposition," in *Foundations of Software Science and Computational Structures (FoSSaCS)*. Berlin, Germany: Springer-Verlag, 2005, pp. 348–363.

[31] M. Silva, E. Teruel, and J. M. Colom, "Linear algebraic and linear programming techniques for the analysis of place/transition net systems," in *Lectures on Petri Nets I: Basic Models*, vol. 1491, W. Reisig and G. Rozenberg, Eds. New York: Springer-Verlag, 1998, pp. 309–373.

[32] J. H. Tracey, "Internal state assignments for asynchronous sequential machines," *IEEE Trans. Electron. Comput.*, vol. EC-15, no. 4, pp. 551–560, Aug. 1966.

[33] S. H. Unger, *Asynchronous Sequential Switching Circuits*. New York: Wiley-Interscience, 1969.

[34] P. Vanbekbergen, "Synthesis of asynchronous control circuits from graph-theoretic specifications," Ph.D. dissertation, Catholic Univ. Leuven, Leuven, Belgium, Sep. 1993.

[35] P. Vanbekbergen, G. Goossens, F. Catthoor, and H. J. D. Man, "Optimized synthesis of asynchronous control circuits from graph-theoretic specifications," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 11, no. 11, pp. 1426–1438, Nov. 1992.

[36] P. Vanbekbergen, B. Lin, G. Goossens, and H. de Man, "A generalized state assignment theory for transformations on signal transition graphs," in *Proc. ICCAD*, Nov. 1992, pp. 112–117.

[37] A. Xie, S. Kim, and P. A. Beerel, "Bounding average time separations of events in stochastic timed Petri nets with choice," in *Proc. Int. Symp. Adv. Res. Asynchronous Circuits Syst.*, Apr. 1999, pp. 94–107.

[38] K. Y. Yun, "Automatic synthesis of extended burst-mode circuits using generalized C-elements," in *Proc. EURO-DAC*, Sep. 1996, pp. 290–295.

Josep Carmona received the M.Sc. and Ph.D. degrees in science from the Universitat Politècnica de Catalunya, Barcelona, Spain, in 1999 and 2004, respectively.

He was a Visiting Scholar with the University of Leiden, Leiden, The Netherlands, in 2003. He is currently a Lecturer with the Department of Software, Universitat Politècnica de Catalunya. His research interests include formal methods and computer-aided designs of very large scale integration systems, with special emphasis on asynchronous circuits, concurrent systems, logic synthesis, and nanocomputing.

Jordi Cortadella (M'88) received the M.S. and Ph.D. degrees in computer science from the Universitat Politècnica de Catalunya, Barcelona, Spain, in 1985 and 1987, respectively.

He was a Visiting Scholar with the University of California, Berkeley, in 1988. He is currently a Professor with the Department of Software, Universitat Politècnica de Catalunya. He is a coauthor of numerous research papers. His research interests include formal methods and computer-aided designs of very large scale integration systems, with special emphasis on asynchronous circuits, concurrent systems, and logic synthesis.

Dr. Cortadella has served on the technical committees of several international conferences on design automation and concurrent systems and has been invited to present tutorials at various conferences. He is a recipient of the Best Paper Awards in the 10th International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC 2004) and the 41st Design Automation Conference (DAC 2004) and of a Distinction for the Promotion of the University Research from the Generalitat de Catalunya in 2003.