Fundamenta Informaticae 34 (2002) 1–23
IOS Press

A structural encoding technique for the synthesis of asynchronous circuits*

Josep Carmona
Computer Architecture Department
Universitat Politècnica de Catalunya
Barcelona, Spain
jcarmona@ac.upc.es

Jordi Cortadella
Software Department
Universitat Politècnica de Catalunya
Barcelona, Spain
jordic@lsi.upc.es

Enric Pastor
Computer Architecture Department
Universitat Politècnica de Catalunya
Barcelona, Spain
enric@ac.upc.es

Abstract. This paper presents a method for the automatic synthesis of asynchronous circuits from Petri net specifications. The method is based on a structural encoding of the system in such a way that a circuit implementation is *always guaranteed*. Moreover, a set of transformations is presented for the subclass of Free-Choice Petri nets that enables the *exploration* of different solutions. The set of transformations is derived from previous work on Petri net synthesis. Both the encoding technique and the set of transformations preserve the property of free-choiceness, thus enabling the use of structural methods for the synthesis of asynchronous circuits. Preliminary experimental

*This work has been partially funded by the Ministry of Science and Technology of Spain under contract TIC 2001-2476, ACiD-WG (IST-1999-29119), a grant by Intel Corporation and CIRIT 2001SGR-00254.

results indicate that the quality of the circuits is comparable to that obtained by methods that require an exhaustive enumeration of the state space.

This novel synthesis method opens the door to the synthesis of large control specifications generated from hardware description languages.

Keywords: Asynchronous circuits, structural synthesis, Complete State Coding, Petri nets.

1. Introduction

In the last few years, there has been an increasing interest in asynchronous circuits. Potential advantages, such as modularity, absence of clock skew problems, average performance and low power, have encouraged many researchers and designers to devote considerable effort towards understanding and proposing techniques for asynchronous circuit design [10].

If some unanimity exists about asynchronous circuits, it is that they are difficult to design. The absence of clock does not allow a discrete abstraction of time and, therefore, the behavior of any signal at any instant can be relevant for the correctness of the circuit. A significant effort has been spent in studying and proposing automatic synthesis techniques that can alleviate the burden of designing asynchronous circuits. This paper focuses on techniques for the synthesis of control circuits.

Currently, there are several academic tools that work at the logic level and attempt to optimize the resulting circuit by using variations of the state-of-the-art Boolean minimization techniques [13, 6, 23]. Given that asynchronous circuits are typically modeled as concurrent systems, the existing synthesis approaches often suffer from the state explosion problem derived from concurrency.

A crucial problem of most asynchronous logic synthesis tools is that they are not always capable of deriving an implementation from the specification. The main reason for that is that some of the implementation properties must be ensured by transforming the specification. And this task is performed automatically by using heuristics that cannot explore the complete space of configurations.

Direct translation methods that do not exploit the power of Boolean minimization have also been proposed [12, 4, 1, 19]. This type of strategies guarantees an implementation by construction, but does not exploit the potential optimizations that can be performed at the logic level. Typically, the size of the obtained circuits is linear on the size of the specification.

There have been few attempts to combine both approaches [27, 16]. However, direct translation methods usually generate circuit structures that cannot be locally transformed to derive succinct representations of the same behavior. For this reason, the results obtained by these methods are comparable to peephole optimizations realized on the original structures.

Nowadays, the knowledge of asynchronous techniques have reached a level of maturity that have enabled some researchers to face the problem of synthesis from Hardware Description Languages (HDLs), such as Verilog [3] or VHDL [30]. This new trend also implies dealing with control circuits that are both *large* and *well-structured*.

Due to the aforementioned state explosion problem, there exist severe limitations on the size of the specifications that can be handled by existing synthesis tools. However, the fact that control specifications derived from HDLs tend to be well-structured opens the door to use techniques that do not require an explicit representation of the state space.

This paper presents some contributions into that direction, with the aim that automatic synthesis techniques based on the presented theoretical results will be proposed in the future. The concurrent model used in this paper is based on Petri nets [22]. The main contribution consists in proposing a set of structural transformations of the specification that *guarantees an implementation* of the behavior without explicitly enumerating the state space of the system. The transformations are proposed for the subclass of Free-choice Petri nets. This subclass seems to be a good trade-off between the expressiveness power required by well-structured control specifications and the methods that can manipulate them without suffering from the size of the state space.

Moreover, the presented transformations preserve the structural properties of the specification, thus enabling the use of logic synthesis techniques that do not require an explicit representation of the state space [26].

The paper is organized as follows. Section 2 describes previous and related work. Section 3 presents basic definitions and background used along the paper. The encoding method and its properties is presented in Section 4. The property-preserving transformations are described in Section 5. Finally, Section 6 illustrates the method with an example and reports some preliminary results.

2. Related work and overview

Signal Transition Graphs (STGs) [28, 5] are interpreted Petri nets used for the specification and synthesis of asynchronous controllers. In STGs, transitions represent rising and falling signal transitions, denoted by positive and negative events (e.g. $a+$, $a-$). Several techniques that circumvent the state explosion problem have been proposed for the synthesis from STGs [29]. However, most of them only work for marked graphs, a very restrictive class of specifications that cannot model choice behaviors [14].

To the best of our knowledge, the only work in this area that has covered the synthesis of specifications with Free-choice Petri nets was presented in [26]. Besides allowing the specification of choice, Free-choice Petri nets also have nice structural properties that enable the use of polynomial algorithms to analyze their behavior [11].

Unfortunately, none of the methods mentioned before has been able to effectively tackle the problem of finding an encoding of the specification that guarantees an implementation. Even the known structural methods working for some subclasses of STGs rely on the fact that heuristics with affordable computational cost will find a solution with high probability [33, 25].

The encoding problem is illustrated in Figure 1. Given a specification (Figure 1(a)), each state of the reachability graph is assigned a binary vector that represents the value of each signal at that state (Figure 1(b)). For a circuit to be derived from the specification, it is required that the value of the signals can uniquely distinguish non-equivalent states. In this example, there are two states that cannot be distinguished by their codes (shadowed in the figure). Solving the state encoding problem is usually

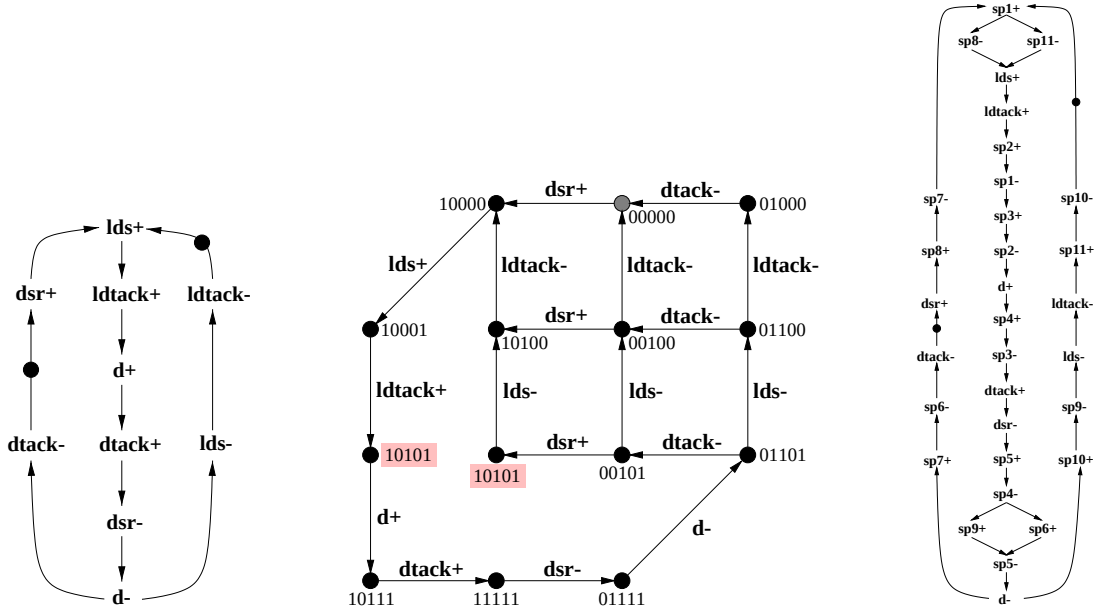


Figure 1 (a) STG, (b) Encoded graph (<dsr,dtack,ldtack,d,lds>), (c) Structurally encoded STG.

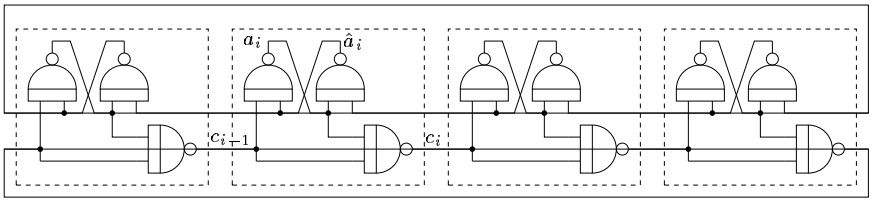
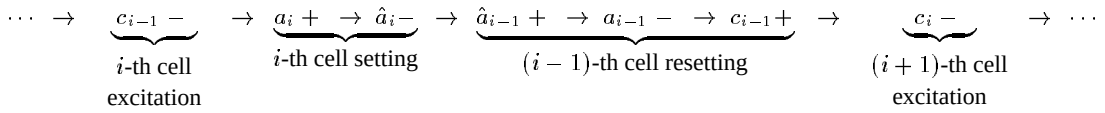


Figure 2. Distributor built from David cells [15].

performed by adding new signals in the specification that preserve implementation properties. Doing so is not an easy task [7].

The method presented in this paper has been inspired on previous work for the direct synthesis of circuits from Petri nets. One of the relevant techniques was proposed in [32], where a set of cells that mimic the token flow in Petri nets was designed. The circuit was built by abutting the cells and producing a structure isomorphic to the Petri net. This type of cells, called David cells, were initially proposed in [9].

Figure 2 depicts a very simple example on how these cells can be abutted to build a distributor that controls the propagation of activities along a ring. The behavior of one of the cells in the distributor can be summarized by the following sequence of events:



In [32], each cell was used to represent the behavior of one of the transitions of the Petri net. The approach presented in this paper is based on encoding the system by inserting a new signal for each place with a behavior similar to a David cell. With such an encoding approach, two goals are achieved:

- A solution for the encoding problem is guaranteed at the expense of over-encoding the states of the system.
- The structural properties of the specification are preserved, thus enabling the use of transformations to optimize the resulting circuit.

In the forthcoming sections, the encoding method and a set of optimizing transformations are discussed.

3. Petri Nets and Signal Transition Graphs

The theory presented in this paper holds for the class of consistent and deterministic Signal Transition Graphs with an underlying Free-choice live and safe Petri net. The necessary definitions to support the theory are next presented.

3.1. Petri Nets

A *Petri Net* (PN) is a 4-tuple $\Sigma = \langle \mathcal{P}, \mathcal{T}, \mathcal{F}, M_0 \rangle$, where $\mathcal{P}$ is the set of places, $\mathcal{T}$ is the set of transitions, $\mathcal{F} \subseteq (\mathcal{P} \times \mathcal{T}) \cup (\mathcal{T} \times \mathcal{P})$ is the flow relation, and M_0 is the initial marking. A marking of a PN is an assignment of a non-negative integer to each place. If k is assigned to place p by marking M (denoted $M(p) = k$), we will say that p is marked with k tokens. Given a node $x \in \mathcal{P} \cup \mathcal{T}$, its pre-set and post-set are denoted by ${}^\bullet x$ and $x^\bullet$ respectively.

A *path* in a PN is a sequence $u_1 \ldots u_r$ of nodes such that $\forall i, 1 \leq i < r : (u_i, u_{i+1}) \in \mathcal{F}$. A path is called *simple* if no node appears more than once on it.

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', denoted by $M[t_1 t_2 \ldots t_n \rangle M'$. A sequence of transitions $t_1 t_2 \ldots t_n$ is a *feasible sequence* if it is firable from M_0. The set of reachable markings from M_0 is denoted by $[M_0\rangle$.

A place in a PN is *redundant* if its elimination does not change the behavior of the net. A PN is *place-irredundant* if it does not have redundant places.

A PN is *live* iff every transition can be infinitely enabled through some feasible sequence of firings from any marking in $[M_0\rangle$. A PN is *safe* if no marking in $[M_0\rangle$ assigns more than one token to any place. A *Free-Choice Petri net* is a PN such that if $(p, t) \in \mathcal{F}$ then ${}^\bullet t \times p^\bullet \subseteq \mathcal{F}$, for every place p [11]. In

Figure 3. (a) Non Free-choice PN, (b) Free-choice PN.

the rest of the paper, we will deal with Free-choice live and safe Petri nets (FCLSPN). An example of non-Free-choice and Free-choice PN is shown in Figure 3. The net in 3(a) is not Free-choice because $(p1, t2) \in \mathcal{F}$ but ${}^\bullet t2 \times p1^\bullet \not\subseteq \mathcal{F}$.

Checking for liveness, safeness and redundant places can be done in polynomial time for Free-choice Petri nets [11].

3.2. Signal Transition Graphs

A Signal Transition Graph (STG) [28] is a triple $\langle \Sigma, A, \Lambda \rangle$, where Σ is a PN, A is a set of signals, partitioned into input signals (A_I), output signals (A_O), and internal signals (A_{INT}), and Λ is the labeling function $\Lambda : \mathcal{T} \rightarrow (A \times \{+, -\}) \cup \{\varepsilon\}$, where all transitions not labeled with the *silent* event (ε) are interpreted as signal changes. The set $A_I \cup A_O$ is called the *observable* set of signals. Rising and falling transitions of a signal $a \in A$ are denoted by $a+$ and $a-$, respectively, while $a*$ denotes a generic rising or falling transition. For the remainder of the paper, we will often use the label of a transition to denote the transition itself.

An example of STG is shown in Figure 1(a). For simplicity, those places that only have one predecessor and one successor transition are not depicted. In that case, the tokens are held on the corresponding arcs.

3.3. Observational equivalence

Milner defined in [20] CCS (Calculus of Communicating Systems), a process algebra for modelling concurrent systems. There, a notion of *unobservable* action (called τ) was defined, representing the fact that a system can 'spontaneously' change its state, without any apparent, or observable, reason. Using this notion one can reason about the equivalence of two systems with respect to their *observable* actions.

The notion of observational equivalence, as defined by Milner in [20], with respect to a set of observable events is relevant in this paper. Informally, two systems are *(weak) observationally equivalent* if their behavior cannot be distinguished by interacting with them. When necessary, we will consider

observational equivalence with respect to input and output signals (not internal), or with respect to all signals. The following definitions assume observational equivalence with respect to all signals.

An STG is *deterministic* if the firing of two different transitions with the same label in a marking $M \in [M_0\rangle$ leads to observational equivalent markings.

A signal is said to be *enabled* in a marking M if there is a marking M' which is observationally equivalent to M and a transition of the signal is enabled in M' (as a particular case, $M = M'$).

3.4. Concurrency and ordering relations

A pair of transitions $t_i, t_j \in \mathcal{T}$ are said to be *concurrent* if there is a marking $M \in [M_0\rangle$ such that $M[t_j t_i\rangle$ and $M[t_i t_j\rangle$. The concept of concurrency can be extended to signals. Two signals a and b are said to be concurrent if there are two transitions with labels $a*$ and $b*$ that are concurrent.

An STG is *non-autoconcurrent* if it does not contain any pair of concurrent transitions of the same signal. An STG satisfies the *consistency* condition if it is non-autoconcurrent and the signal changes in every feasible sequence of signal transitions alternate. This last condition restricts the feasible sequences: the change $0 \rightarrow 1$ ($1 \rightarrow 0$) can only be followed by the change $1 \rightarrow 0$ ($0 \rightarrow 1$) for each signal appearing in a feasible sequence.

Let $R \subseteq [M_0\rangle$ be the set of markings where transition t_i is enabled. Transition t_j *triggers* transition t_i if there exists a reachable marking M such that $M[t_j\rangle M'$, $M \notin R$ and $M' \in R$.

Transition t_j *disables* transition t_i if there exists a reachable marking M enabling both t_i and t_j, but in the marking M' such that $M[t_j\rangle M'$, t_i is not enabled.

3.5. Encoding

Each marking of an STG is encoded with a *binary vector* of signal values by means of a labeling function $\lambda : [M_0\rangle \rightarrow \{0,1\}^{|A|}$. All markings must be *consistently encoded* by λ, i.e. no marking M can have an enabled rising (falling) transition $a+$ ($a-$) if $\lambda(M)_a = 1$ ($\lambda(M)_a = 0$).

Figure 1(b) depicts the set of reachable states derived from the STG in Figure 1(a), with the corresponding encoding.

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 STG in Figure 1(a) does not satisfy the CSC property, since there are two different markings with the code 10101, but in one marking the output transition $d+$ is enabled while in the other $lds-$ is enabled.

A more restrictive property, called *unique state coding* (USC), holds if all reachable markings are assigned a unique binary code, i.e., $\forall M_1, M_2 \in [M_0\rangle : M_1 \not\approx M_2 \Rightarrow \lambda(M_1) \neq \lambda(M_2)$, where $\approx$ denotes observational equivalence.

The CSC property is a necessary condition for the correct implementation of an STG specification. When the CSC condition holds, the events that the circuit must produce at each reachable state are uniquely determined by the binary code of the state itself.

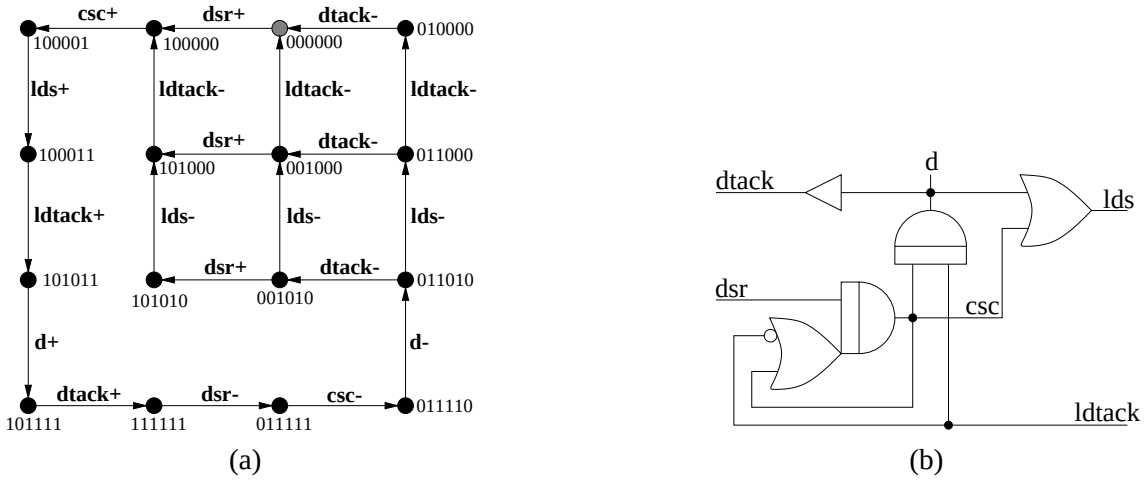


Figure 4. (a) Signal insertion forcing CSC, (b) Final circuit.

In the last decade, there have been several approaches for the development of automatic encoding algorithms for STGs [18, 31, 33, 24, 7]. The way the encoding is performed is usually transforming the specification, by adding new signals that aim at disambiguating the conflicting states. Figure 4(a) shows a possible signal insertion that solves the conflict of the specification of Figure 1. The insertion of the new signal *csc* disambiguates the encoding conflict.

3.6. Synthesis of speed-independent circuits

Here, we briefly sketch how a circuit can be derived from an STG. This theory is valid for the class of *speed-independent* circuits, which are correct when assuming that all components of the circuit can have any delay [21].

For an STG S to be correctly implemented by an speed-independent circuit, four conditions must hold [17]:

1. the set of reachable states of S must be finite (*boundedness*),
2. S must fulfill the CSC property,
3. function λ must consistently encode the reachable markings of S (*consistency*),
4. for any pair of signals x and y such that x disables y it implies that x and y are input signals (*output persistency*).

If we call $a_1, \ldots, a_n$ the signals of the circuit, each non-input signal x can be implemented by a gate that realizes a logic function f_x. The logic function is defined for each binary vector $v \in \{0,1\}^n$ as follows:

$$f_x(v) = \begin{cases} 1 & \text{if } \exists M : \lambda(M) = v \ \wedge \ (\text{some } x+ \text{ enabled in } M \vee (\lambda(M)_x = 1 \wedge \text{no } x- \text{ enabled in } M)) \\ 0 & \text{if } \exists M : \lambda(M) = v \ \wedge \ (\text{some } x- \text{ enabled in } M \vee (\lambda(M)_x = 0 \wedge \text{no } x+ \text{ enabled in } M)) \\ - & \text{if } \nexists M : \lambda(M) = v \end{cases}$$

In case the CSC property does not hold, the previous definition is ambiguous, since a binary vector could be found for which there are two different markings that would make f_x equal to 0 and 1 simultaneously.

The previous function is incompletely specified. For those vectors in which $f_x(v) = -$, the function may take any value, since those vectors will never appear in any reachable state of the system. This set of vectors defines the *don't care* set of the function, which is extremely important for an efficient Boolean minimization.

The specification of Figure 4(a) fulfills the speed-independent conditions for correct synthesis. The circuit synthesized is shown in Figure 4(b).

3.7. IO-STGs

The *I/O interface* [8] of a reactive system describes those causality relations that must be preserved in the protocol between the system and the environment where the system operates. In Section 4.2 a transformation applied to STG specifications will be presented that guarantees the CSC property on the transformed STG. For the preservation of the I/O interface, this transformation can only be applied to a restricted class of specifications: the IO-STG class contains those STGs fulfilling both that no input signal transition triggers another, and that the transitions in the post-set of a *choice* place are all input signal transitions. More formally:

Definition 3.1. An IO-STG is a Free-Choice STG where the following conditions hold:

1. $\forall a_i* \text{ with } a \in A_I : \forall b_j* : (b_j* \in (a_i*^\bullet)^\bullet \Rightarrow b \notin A_I)$.
2. $\forall p \in \mathcal{P} : (|p^\bullet| \geq 2 \Rightarrow \forall b_j* \in p^\bullet : b \in A_I)$.

4. Structural Encoding

This section presents a transformation applied to STGs. The features of this transformation are the following:

- It guarantees the USC property.
- It preserves free-choiceness.
- It preserves consistency, liveness, safeness and observational equivalence with respect to the input and output signals.
- It has linear complexity on the size of the STG.

This is the first method that *guarantees* a solution for the encoding problem and tackles the problem in linear complexity for the class of FCLSPNs. The transformation is based on the insertion of a signal for each place of the STG that mimics the token flow on that place.

Althought the encoding transformation does not require the net to be Free choice, it preserves this property. This is important in our framework to enable the use of structural methods for synthesis.

1. Create the $silent$ transitions ε_1 and ε_2.
2. For each place $p \in {}^\bullet t$, create a new transition with label $sp-$ and insert new arcs and places for creating a simple path from ε_1 to ε_2, passing through $sp-$.
3. For each place $q \in t^\bullet$, substitute the arc (t, q) by the arc (ε_2, q), create a new transition labeled as $sq+$ and insert new arcs and places for creating a simple path from t to ε_1, passing through $sq+$.

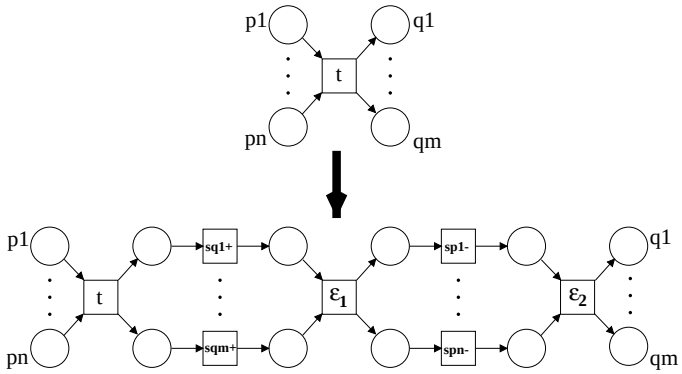


Figure 5. Transformation rule for each transition $t \in \mathcal{T}$.

The transformations will be presented as a rule to be applied to the transitions of the STG. Before the application of the Structural Encoding, the set of signals of the STG has been augmented with one signal sp for each place p of the STG. In order to simplify the presentation of the rules and the corresponding proofs, we will use silent transitions on the definition of the rules.

4.1. Encoding transformation

Let $S = \langle\langle \mathcal{P}, \mathcal{T}, \mathcal{F}, M_0\rangle, A, \Lambda\rangle$ be an STG with underlying FCLSPN. The Structural Encoding of S derives the STG $Enc(S)$ in which a new internal signal sp has been created for each place $p \in \mathcal{P}$, and the transformation rule described in Figure 5 has been applied to each transition $t \in \mathcal{T}$. The new transitions appearing in $Enc(S)$, labelled with $sp*$, will be called *E-transitions* along the paper.

Let us now prove properties on $Enc(S)$.

Proposition 4.1. *$Enc(S)$ is free-choice.*

Proof:
Every new place p appearing in $Enc(S)$ has $|{}^\bullet p| = |p^\bullet| = 1$ by construction. For each place p (transition t) of S, the set $p^\bullet$ (${}^\bullet t$) is identical both in S and $Enc(S)$. Given that S is free-choice, $Enc(S)$ is also free-choice.

□

Proposition 4.2. *$Enc(S)$ is live, safe and is observationally equivalent to S with respect to the input and output signals.*

Proof:
The transformation for structural encoding is a trivial combination of a set of transformations proposed by Berthelot that preserve liveness, safeness and home marking [2]. These transformations also preserve the behavior condition: each conflict resolution in $Enc(S)$ is performed by some observable transition, i.e. for every transition $x*$ and place p such that $p \in {}^\bullet x*$ and $p^\bullet > 1$ then $x \in A_I \cup A_O$.

From the behavior condition, it immediately follows that observational equivalence is also preserved. □

Proposition 4.3. *$Enc(S)$ is consistent.*

Proof:
Given that the observational equivalence is preserved, consistency directly holds for the signals already in S. It only remains to prove that it also holds for the E-transitions of the new inserted signals.

By construction, the new sp and sq signals mimic the token flow in places. Given that the dynamic behavior corresponds to a *safe* PN, no more than two consecutive rising or falling transitions can occur for these signals. □

Lemma 4.1. *Let R be the new set of places inserted in S for constructing $Enc(S)$. Every feasible complementary set between two reachable markings M and M' of $Enc(S)$ satisfies the equality $M|_R = M'|_R$.*

Proof:
Figure 6 depicts a fragment of $Enc(S)$ that results from applying the transformation rule to a transition with $p_1 \ldots p_n$, and $q_1 \ldots q_m$ as predecessor and successor places, respectively. Without loss of generality, we will assume that the label of the transition is $x+$, and that place q_i has one successor transition with label $y+$. By definition, if a feasible complementary set exists between M and M' then $\lambda(M) = \lambda(M')$. With two exceptions that will be discussed later, the marking of the new places inserted (places a, b, c and d in Figure 6) can be uniquely determined as follows:

$$
\begin{aligned}
M(a_i) = 1 \quad &\Leftrightarrow \quad sq_i = 0 \ \wedge \ sp_1 = \cdots = sp_n = 1 \ \wedge \ x = 1 \\
M(b_i) = 1 \quad &\Leftrightarrow \quad sq_i = 1 \ \wedge \ sp_1 = \cdots = sp_n = 1 \ \wedge \ x = 1 \\
M(c_i) = 1 \quad &\Leftrightarrow \quad sp_i = 1 \ \wedge \ sq_1 = \cdots = sq_m = 1 \ \wedge \ x = 1 \\
M(d_i) = 1 \quad &\Leftrightarrow \quad sp_i = 0 \ \wedge \ sq_1 = \cdots = sq_m = 1 \ \wedge \ x = 1
\end{aligned}
$$

When defining the previous equations, it is important to use the fact that the STG is safe and consistent. We will only prove the equality for $M(a_i)$. The other equalities can be proved in a similar way.

$\Rightarrow$

$M(a_i) = 1$ implies $sq_i = 0$, since sq_i+ is enabled. Otherwise the STG would not be consistent. $M(a_i) = 1$ also implies $sp_1 = \cdots = sp_n = 1$, since the liveness and safeness of the STG imply that ε_1 has not fired after $x+$ has fired. Therefore, none of the sp_i- transitions has fired yet, while all sp_i+ transitions already fired before $x+$. Finally, $M(a_i) = 1$ clearly implies $x = 1$.

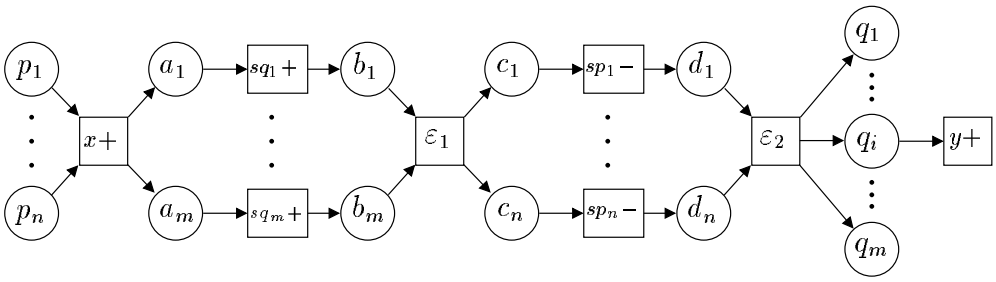


Figure 6. Place encoding to guarantee USC.

$\Leftarrow$

By the consistency of signal x, the only markings in which $sp_1 = \cdots = sp_n = 1$ and $x = 1$ correspond to markings in which some tokens are held in the places after $x+$ but before $sp_1 - \ldots sp_n-$. The fact that $sq_i = 0$ implies that place a_i has a token.

As mentioned before, there are two exceptions in which the binary code does not uniquely identify the marking in the new places inserted. One exception corresponds to the submarkings in which $M(b_1) = \cdots = M(b_m) = 1$ and $M(c_1) = \cdots = M(c_n) = 1$, respectively. These submarkings are only separated by a silent transition, ε_1, that makes them observationally equivalent. The other exception corresponds to the submarkings separated by ε_2.

Finally, given that $\lambda(M) = \lambda(M')$ we can conclude that the previous equations also hold for M', and therefore the marking in R is identical both in M and M'. □

Lemma 4.2. *Let q_i be a place of S, and T_c be a feasible complementary set between two reachable markings M and M' of $Enc(S)$. Then, $M(q_i) = 1 \Rightarrow M'(q_i) = 1$.*

Proof:
Without loss of generality, let q_i be the one of Figure 6. Assume that $M(q_i) = 1$. If no transition in $q_i^\bullet$ belongs to T_c then the claim trivially holds.

The initial situation is depicted in Figure 7.

Asumme, without loss of generality (due to the free-choiceness of $Enc(S)$), that $y+ \in T_c$, and let M'' be the marking reached after firing $y+$. From M'' it is possible to fire the set of E-transitions wich result from the encoding of $y+$. One transition of this set is sq_i-, but note that $\lambda(M)_{sq_i} = 1$. Two situations can happen:

1. $sq_i- \in T_c$: then at least a transition sq_i+ belongs to T_c. But every copy of an sq_i+-transition is either in $(x+^\bullet)^\bullet$ or in $(z+^\bullet)^\bullet$, where $z+$ is another transition such that ${}^\bullet z+ = {}^\bullet x+$[1]. Assume that the transition belonging to T_c is the one in $(z+^\bullet)^\bullet$. The safenes of $Enc(S)$ ensures that the set of places from the encoding of $z+$ is unmarked in M, because otherwise there is a marking reachable from M having two tokens on q_i. But Lemma 4.1 ensures that the marking in the new places inserted for encoding $z+$ is the same both in M and M', and therefore every E-transition from the encoding of $z+$ belongs to T_c, adding again a token to q_i.

[1] In the simplest case, $z+ = x+$.

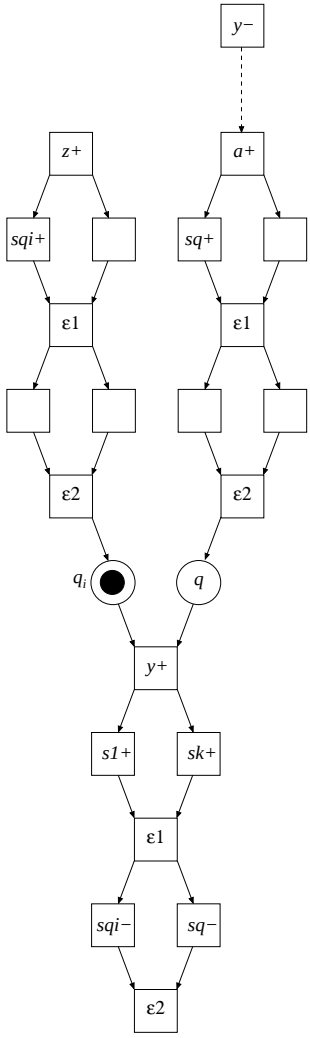


Figure 7. Initial situation for proof of Lemma 4.2.

2. $sq_i- \notin T_c$: every adjacent transition $y-$ such that $y- \in next(y+)$ can not appear in T_c after $y+$ because transition ε_2 from the encoding of $y+$ does not belongs to T_c, and then given that $Enc(S)$ is consistent no sequence of transitions in T_c can enable $y-$ after the firing of $y+$[2]. Therefore $y-$ appears before of $y+$ in T_c. Again, the consistency of $Enc(S)$ implies that there exists a place q such that $q \in {}^\bullet y+$, $M(q) = 0$ and some $sq+$ transition in the path between $y-$ and $y+$ must be fired in order to put a token in q. Then there must be in T_c an $sq-$ transition, but the actual situation is not possible in $Enc(S)$, because:

[2]Estic usant les dues condicicions de consistencia per garantir-ho: switchover-correctness per a control.lar que hi ha una unica forma de disparar les dues transicions i non-autoconcurrency per garantir que si la $y+$ no passa els tokens cap endavant, aleshores no pot ser que la $y-$ s'habiliti posteriorment.

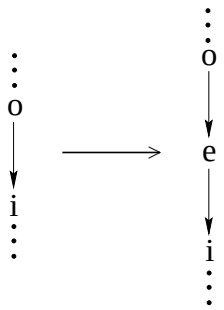


Figure 8. Tranformation not preserving the I/O interface.

- The positive E-transitions from the encoding of $y+$ (let s_1+, ..., s_k+) must be in T_c in order to enable $sq-$,
- Lemma 4.1 ensures that the same marking exists both in M and M' with respect to the set of places from the encoding of $y+$. Given that $sq_i- \notin T_c$ implies that transition ε_2 from the encoding of $y+$ is not in T_c, and then in M (M') some of this places are marked.
- And then the consistency of the E-signals inserted s_1, ..., s_k in $Enc(S)$ implies that the set of negative transitions s_1-, ..., s_k- can not be in T_c, because otherwise them can be autoconcurrent. But then T_c is not a complementary set.

□

Proposition 4.4. *Enc(S) has the USC property.*

Proof:
This follows from Lemmas 4.1 and 4.2, together with the boundedness of $Enc(S)$.

□

4.2. Preserving the Input/Output Interface

Preserving the observational equivalence with respect to the input and output signals of the specification is not sufficient to guarantee a correct implementation of a system. When one wants to implement a module of a system as a circuit, the input/output interface for that module is typically fixed a priori. From the point of view of the circuit, the environment can be considered as another module with mirrored signals (input and outputs of the circuit are outputs and inputs of the environment, respectively).

Since the environment must be considered as an already implemented system that cannot change its interface, the causality relations between the outputs of the circuit and the inputs of the environment must be preserved. In practice, this means that if the firing of an output signal may enable an input signal, then this causality must be preserved along any transformation of the specification.

For a transformation to preserve the I/O interface, it must fulfill the following conditions [8]:

1. The set of triggering events of every input signal transition is preserved.
2. The initial state is preserved with respect to the observable signals.

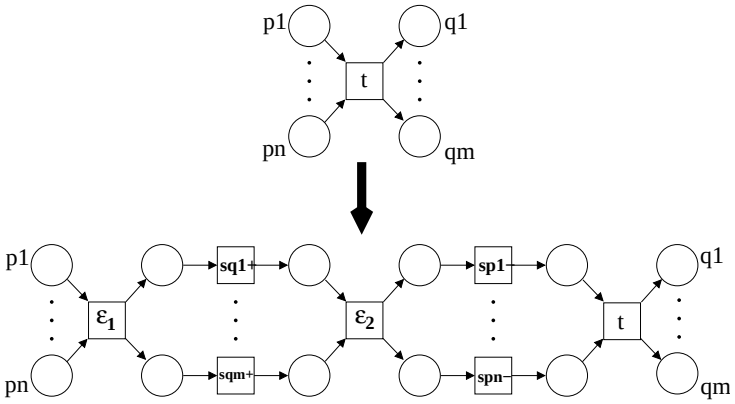


Figure 9. Transformation rule for non-input signals to preserve the I/O interface.

Figure 8 shows a transformation (insertion of the internal event e) not preserving the I/O interface. In the Figure, i is an input event whereas o is output.

The encoding technique presented in Section 4.1 does not preserve the I/O interface, because when the transformation rule is applied to an output signal transition being a trigger of an input signal transition, internal events (induced by the intermediate places of the initial STG) are inserted delaying the input signal transition, and therefore there is a violation of condition 1.

However, the encoding technique presented in Section 4.1 can be modified to preserve the I/O interface. The refinement presented below is only valid for the IO-STG class. In that class, the transformation rule shown in Figure 9 can be applied to any transition of a non-input signal. For input transitions, the previous transformation presented in Figure 5 is applied. This refined encoding technique is called *IO-Enc*(S).

Note that the two transformations only differ on the location of the E-transitions. For non-input signals, the E-transitions precede the transformed transition. In this way, the creation of new causality relations from E-transitions to input transitions is avoided and, thus, I/O interface preserved.

Proposition 4.5. *Let S be an IO-STG with underlying FCLSPN and inititial home marking. IO-Enc(S) preserves the I/O interface with respect to S.*

Proof:
For each input signal transition i, the sets ${}^\bullet i$ and ${}^\bullet({}^\bullet i)$ are preserved in *IO-Enc*(S), and therefore the set of events triggering i is not modified.

IO-Enc(S) and S contain the same set of places marked. Let i be an input signal transition enabled in the initial state of S. Clearly, the previous paragraph implies that i is also enabled in the initial state of *IO-Enc*(S). Let o be an output signal transition enabled in the initial state of S, with ${}^\bullet o = \{p_1, ..., p_n\}$. The transformation rule of Figure 9 applied to o introduces a subnet between $\{p_1, ..., p_n\}$ and o, such that ${}^\bullet \varepsilon_1 = \{p_1, ..., p_n\}$. The liveness of *IO-Enc*(S) implies that from the initial state, a feasible sequence of internal transitions exists enabling o. □

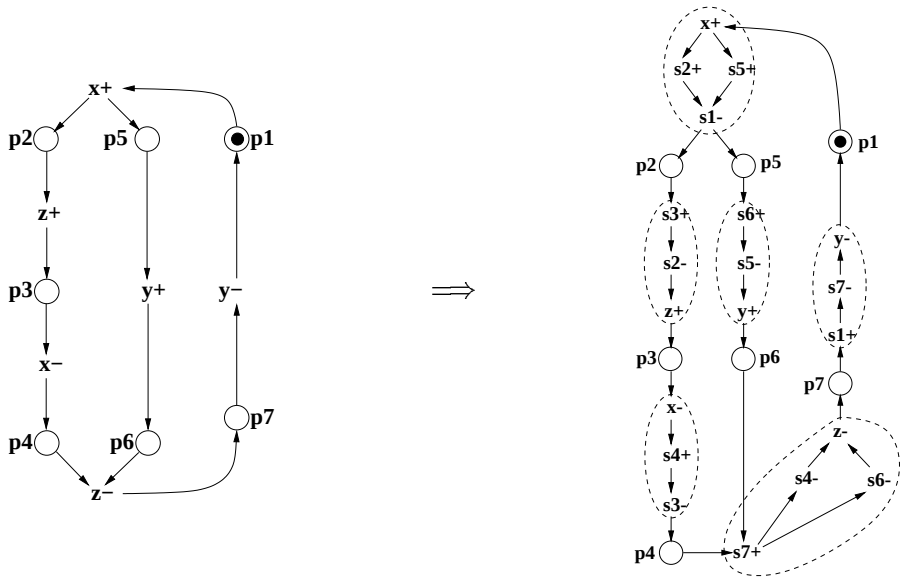


Figure 10. Structural encoding (*x* is input and *y* and *z* are outputs).

The proofs for preserving free-choiceness, liveness, safeness, consistency, observational equivalence and ensuring USC are similar to those presented in the previous section when applied to the class of IO-STGs.

Moreover, the *IO-Enc* technique also preserves the speed-independent conditions.

Proposition 4.6. *Let S be an IO-STG with underlying FCLSPN fulfilling the speed-independent conditions. IO-Enc(S) fulfills the speed-independent conditions.*

Proof:
The first three conditions for correct SI implementation (see Section 3.6) are guaranteed by the previous propositions. Every new place p of *IO-Enc*(S) satisfies $|p^\bullet| = 1$. Every place p of *IO-Enc*(S) such that $|p^\bullet| > 1$ has the same postset both in S and *IO-Enc*(S), and therefore the output persistency of S ensures that no output signal transitions exists in $p^\bullet$. In conclusion, when a signal transition t is enabled and is not an input, every place $p \in {}^\bullet t$ satisfies $|p^\bullet| = 1$, and then t can not be disabled. □

Figure 10 depicts an example of the structural encoding by applying the transformations presented in this section.

5. Design space exploration

Even though the encoding method previously presented guarantees an implementation of the system, the insertion of an internal signal for each place may be too costly, in size and performance, for the final circuit.

This section presents a kit of structural transformations that aim at the exploration of the design space. Behind these transformations, we assume to have a synthesis framework that can evaluate the cost of the implementation in polynomial complexity on the size of the specification [26]. This framework

is also capable of detecting CSC conflicts with low computational cost [25]. The transformations are mainly meant to deal with the new inserted E-transitions. In this context, the strategy used in the search for the optimal is to apply, at each stage of the exploration, the transformation that heuristically improves the actual implementation and does not introduce new CSC conflicts.

The kit of transformations we present are not new in the literature. They have been proposed by other authors (see [2, 22, 11]) or can be obtained by combining several of those transformations. The ones that have been used in this work are next described. All of them preserve the relevant properties of the STGs required for this work.

Concurrency reduction. Given two concurrent transitions, t_i and t_j, such that ${}^\bullet({}^\bullet t_i) \cap {}^\bullet({}^\bullet t_j) \neq \emptyset$, concurrency is reduced by including two places that force an alternation on the firing of the two transitions (see Figure 11(a)).

Increase of concurrency. Given two ordered transitions, t_i and t_j, such that $t_i^\bullet = {}^\bullet t_j = \{p\}$, two parallel branches are created so that they can be fired concurrently. This transformation can be obtained by combining some of the ones presented in [22] (initially proposed in [2]). The transformation is shown in Figure 11(b).

Elimination of signal. Given one of the internal signals of the STG, it can be eliminated by changing the label of all transitions of that signal and making them silent (ε). This transformation is only accepted when the removal of the signal does not create CSC conflicts in the specification.

Elimination of silent transitions. Some of the transformations may insert silent transitions in the specification. By removing them, the size of specification can be reduced and the synthesis algorithms sped-up. However, the elimination the transition requires the substitution of $n + m$ places (predecessor and successor) by $n \cdot m$ places (see Figure 11(c)). Heuristics can be used to determine when the elimination can be useful.

Elimination of redundant places. Some of the places may be eliminated without changing the behavior of the net. Linear programming techniques can be used to determine when places are redundant [2, 11].

On top of this kit of transformations, a search engine is expected to explore the design space. Greedy heuristics or optimization techniques such as simulated annealing, genetic algorithms or tabu search can be used to explore different configurations. All these techniques require a fast estimation of the cost of the explored configurations. The cost function can be efficiently supported by the polynomial algorithms that can be typically used to manipulate free-choice Petri nets.

6. Example and experimental results

The transformations presented in this paper have been applied to well-known specifications from the literature of asynchronous circuit design.

Currently, no search engine is still available to apply the transformations automatically. Instead, they are applied *mechanically* with the intervention of the designer, who at each step, chooses a transformation that intuitively leads to a better implementation.

In more detail, the synthesis strategy consists of the following steps:

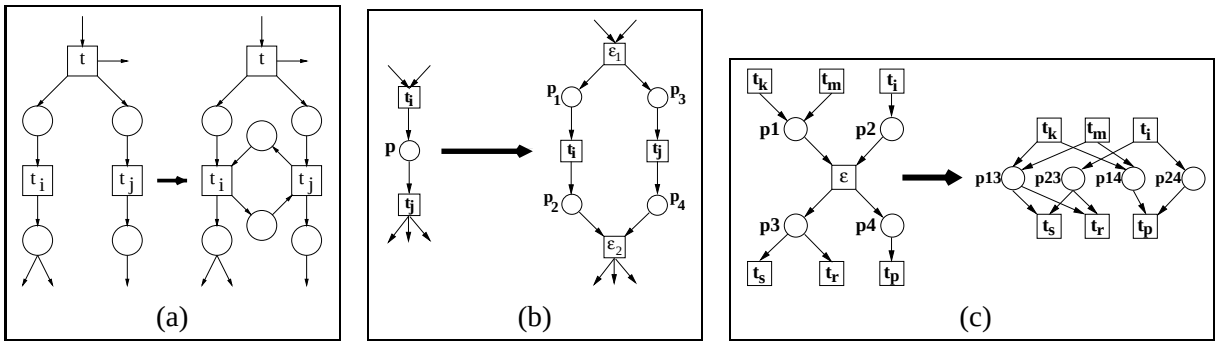


Figure 11. (a) Concurrency reduction, (b) increase of concurrency, (c) transition elimination.

1. Apply the encoding transformations on all transitions, as explained in Section 4.
2. Iteratively and greedily apply the transformation *elimination of signal* to all internal signals as far as no CSC conflicts appear. In general, some of the internal signals will remain in the specification.
3. Iteratively apply transformations and evaluate the implementation cost of the new specification. Accept any transformation that produces a cost improvement. The cost is evaluated as the number of literals of the Boolean equations implementing the circuit.
4. Stop when no transformation can be applied to improve the cost of the circuit.

Given that the previous method has not been automated yet, the application of the transformations at each step has not been done in an exhaustive manner. Structural methods are used for checking CSC and deriving boolean equations [26].

The results have been compared with those obtained by the tool petrify [6], that does an explicit enumeration of the state space, thus suffering from the state explosion problem.

6.1. A case study: *adfast*

This example corresponds to the specification of an *analog-to-digital fast converter* with three input signals (Da, La and Za) and three output signals (Dr, Lr and Zr). The specification is shown in Figure 12(a). This STG does not have the CSC property. The tool petrify automatically inserts two signals to solve CSC conflicts.

Figure 12(d) shows the STG after being transformed by the structural encoding rules. The new internal signals $s_0 \ldots s_{14}$ correspond to the 15 places in the initial specification.

From the STG in Figure 12(d), internal signals are greedily removed until CSC conflicts appear. The resulting STG and the corresponding equations (35 literals) are shown in Figure 13(a). Figure 13(b) reports one of the intermediate solutions (31 literals) explored after obtaining the solution in Figure 13(a).

Figures 12(b) and 12(c) depict the final STG, the Boolean equations and the circuit after applying the transformations and doing logic synthesis. This solution, which has been obtained mechanically, is identical to the one generated by petrify.

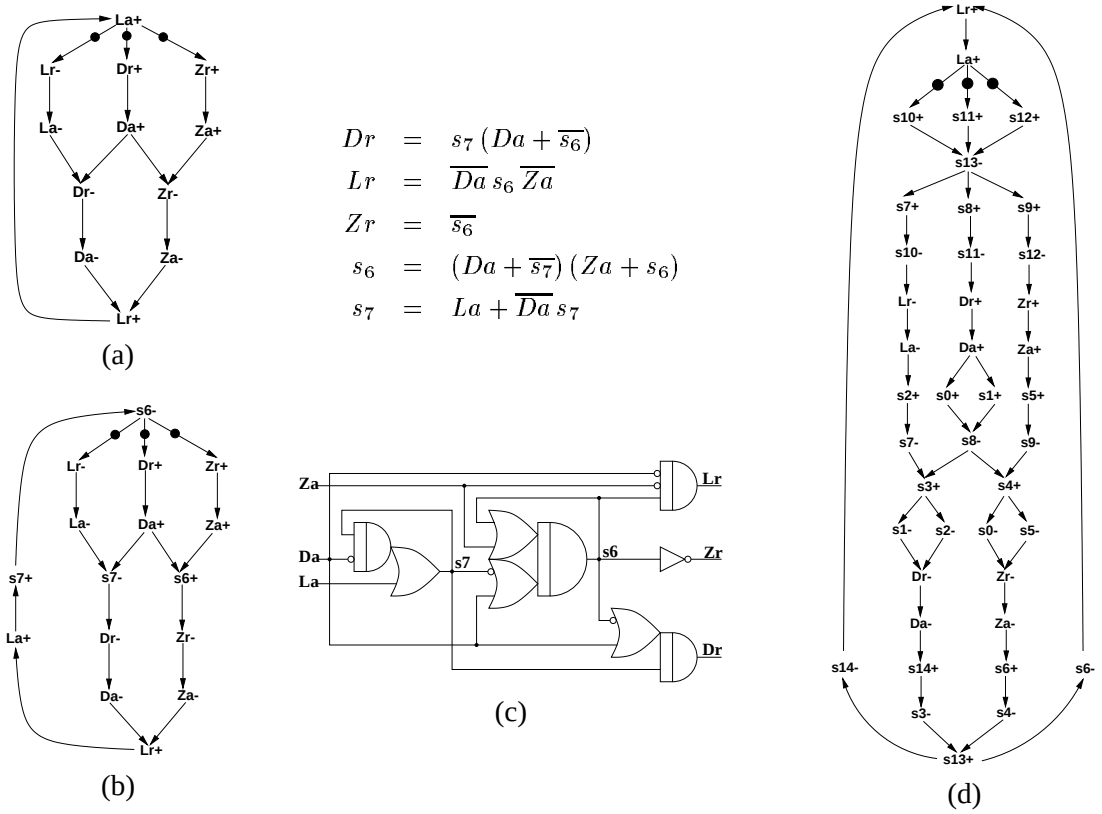


Figure 12. Case study: adfast

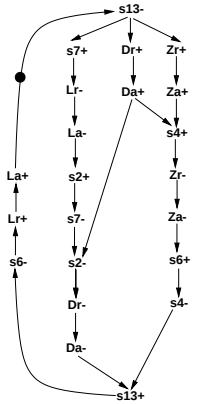
6.2. Experimental results

The synthesis strategy described above has been applied to a set of benchmarks. Initially, none of the specifications had the CSC property. The results are reported in Table 1.

The columns labeled with *"petrify"* indicate the characteristics of the circuit obtained by the tool petrify. The number of inserted signals to solve CSC conflicts and the number of literals of the Boolean equations are reported.

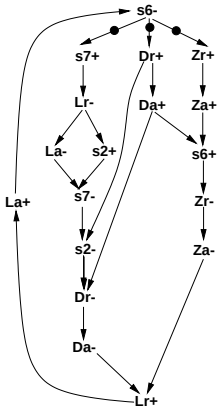
The columns labeled with *"struct. encoding"* report the characteristics of the circuit after having applied steps 1 (encoding) and 2 (elimination of internal signals) of the synthesis strategy. It is interesting to observe that the number of signals required to solve CSC conflicts when using the "local" encoding provided by the places is significantly larger than the number of signals required when "global" encoding methods are used.

The results of the final circuit, after having explored the design space with the set of transformations, are reported in the columns labeled *"str. enc. + optim."*. It can be observed that the quality of the solution can be highly improved by playing with the concurrency of the internal signals. In many cases, the obtained result is the same as the one generated by petrify. In other cases, the results are similar but with more internal signals than the ones inserted by petrify(e.g. master-read2, duplicator). This corroborates a known fact that states that the reduction of internal signals does not always imply an



$$
\begin{aligned}
Dr &= \overline{s_{13}}\, La + s_2 + s_7 \\
Lr &= \overline{s_7}\, La + s_{13}\, \overline{s_6} \\
Zr &= \overline{s_{13}}\, \overline{s_6}\, \overline{s_4} \\
s_2 &= \overline{Da}\, s_2 + s_7\, \overline{La} \\
s_4 &= Za\, (\overline{Lr}\, \overline{s_2}\, \overline{s_7} + Da) + s_4\, \overline{s_6} \\
s_6 &= \overline{s_{13}}\, s_6 + \overline{Za}\, s_4 \\
s_7 &= \overline{s_{13}}\, Lr + \overline{s_2}\, s_7 \\
s_{13} &= \overline{s_4}\, s_6\, \overline{Da} + s_{13}\, \overline{La}
\end{aligned}
$$

(a)



$$
\begin{aligned}
Dr &= La\, (\overline{s_6} + Dr) + \overline{Da}\, Dr + s_2 + s_7 \\
Lr &= \overline{Da}\, s_6\, \overline{Za} + Lr\, \overline{s_7} \\
Zr &= \overline{s_6} \\
s_2 &= \overline{Dr}\, s_2 + \overline{Lr}\, s_7 \\
s_6 &= Da\, (s_6 + Za) + \overline{La}\, (\overline{Dr}\, Za\, \overline{s_2}\, \overline{s_7} + s_6) \\
s_7 &= La\, (\overline{s_6} + Da) + \overline{s_2}\, s_7
\end{aligned}
$$

(b)

Figure 13. Intermediate solutions in the design space.

improvement on the quality of the circuit.

The most important fact that can be deduced from this table is that the method proposed in this paper can compete with the existing synthesis tools. Moreover, for the class of FCLSPNs, this method can guarantee and produce an implementation in extremely low CPU times[3].

7. Conclusions

Methods for the synthesis of systems whose complexity does not depend on the size of the state space are crucial to face the design of complex asynchronous circuits.

This paper has presented an approach for the synthesis of asynchronous controllers from STGs. The main features of the method are: (1) an implementation is guaranteed and (2) the complexity of the method is polynomial on the size of the specification.

[3]The lack of automation in the application of the transformations did not allow a fair report of CPU times. However, this fact became evident in previous works in this area [26]

		petrify		*struct. encoding*		*str. enc. + optim.*	
benchmark	states	#CSC	lit.	#CSC	lit.	#CSC	lit.
adfast	44	2	14	5	35	2	14
vme-fc-read	14	1	8	2	14	1	8
nak-pa	56	1	18	3	35	1	18
m-read1	1882	1	38	2	43	1	40
m-read2	8932	8	68	13	95	10	70
duplicator	20	2	18	5	36	3	18
mmu	174	3	29	7	53	3	34
seq8	36	4	47	22	147	4	47

Table 1. Experimental results.

This work is a first step towards a complete automation of the design flow through the exploration of multiple configurations that preserve some equivalence with the original specification. This exploration should allow to find good trade-offs between the size of the implementation and its performance.

As future work, it is expected to integrate all the algorithms presented in a framework for the synthesis of asynchronous circuits, specified in an HDL.

References

[1] Berkel, K. v.: *Handshake Circuits: an Asynchronous Architecture for VLSI Programming*, vol. 5 of *International Series on Parallel Computation*, Cambridge University Press, 1993.

[2] Berthelot, G.: Checking Properties of Nets Using Transformations, *Advances in Petri Nets 1985* (G. Rozenberg, Ed.), 222, Springer-Verlag, 1986.

[3] Blunno, I., Lavagno, L.: Automated synthesis of micro-pipelines from behavioral Verilog HDL, *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, IEEE Computer Society Press, April 2000.

[4] Burns, S. M., Martin, A. J.: Syntax-directed Translation of Concurrent Programs into Self-timed Circuits, *Advanced Research in VLSI* (J. Allen, F. Leighton, Eds.), MIT Press, 1988.

[5] Chu, T.-A., Glasser, L. A.: Synthesis of Self-timed Control Circuits form Graphs: An Example, *Proc. International Conf. Computer Design (ICCD)*, IEEE Computer Society Press, 1986.

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

[7] Cortadella, J., Kishinevsky, M., Kondratyev, A., Lavagno, L., Yakovlev, A.: A Region-Based Theory for State Assignment in Speed-Independent Circuits, *IEEE Transactions on Computer-Aided Design*, **16**(8), August 1997, 793–812.

[8] Cortadella, J., Kishinevsky, M., Kondratyev, A., Lavagno, L., Yakovlev, A.: Automatic handshake expansion and reshuffling using concurrency reduction, *Proc. of the Workshop Hardware Design*

and Petri Nets (within the International Conference on Application and Theory of Petri Nets), June 1998.

[9] David, R.: Modular design of asynchronous circuits defined by graphs, *IEEE Transactions on Computers*, **26**(8), August 1977, 727–737.

[10] Davis, A., Nowick, S. M.: An Introduction to Asynchronous Circuit Design, in: *The Encyclopedia of Computer Science and Technology* (A. Kent, J. G. Williams, Eds.), vol. 38, Marcel Dekker, New York, February 1998.

[11] Desel, J., Esparza, J.: *Free Choice Petri Nets*, Cambridge University Press, Cambridge, Great Britain, 1995.

[12] Ebergen, J. C.: *Translating Programs into Delay-Insensitive Circuits*, Ph.D. Thesis, Dept. of Math. and C.S., Eindhoven Univ. of Technology, 1987.

[13] Fuhrer, R. M.: *Sequential optimization of asynchronous and synchronous finite-state machines*, Ph.D. Thesis, Columbia University, NY, 1999.

[14] Jung, S. T., Jhon, C. S.: Direct Synthesis of Efficient Speed-independent Circuits from Deterministic Signal Transition Graphs, *Proc. International Symposium on Circuits and Systems*, June 1994.

[15] Kishinevsky, M., Kondratyev, A., Taubin, A., Varshavsky, V.: *Concurrent Hardware: The Theory and Practice of Self-Timed Design*, Series in Parallel Computing, John Wiley & Sons, 1994.

[16] Kolks, T., Vercauteren, S., Lin, B.: Control Resynthesis for Control-Dominated Asynchronous Designs, *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, March 1996.

[17] Kondratyev, A., Kishinevsky, M., Taubin, A., Cortadella, J., Lavagno, L.: The use of Petri nets for the design and verification of asynchronous circuits and systems, *Journal of Circuits Systems and Computers*, **8**(1), 1998, 67–118.

[18] Lavagno, L., Moon, C., Brayton, R., Sangiovanni-Vincentelli, A.: Solving the state assignment problem for signal transition graphs, *Proc. ACM/IEEE Design Automation Conference*, IEEE Computer Society Press, June 1992.

[19] Leung, S. C., Li, H. F.: A Syntax-Directed Translation for the Synthesis of Delay-Insensitive Circuits, *IEEE Transactions on VLSI Systems*, **2**(2), June 1994, 196–210.

[20] Milner, R.: A calculus of communication systems, *Lecture Notes in Computer Science*, 92, Springer-Verlag, 1980.

[21] Muller, D. E., Bartky, W. S.: A Theory of Asynchronous Circuits, *Proceedings of an International Symposium on the Theory of Switching*, Harvard University Press, April 1959.

[22] Murata, T.: Petri Nets: Properties, Analysis and Applications, *Proceedings of the IEEE*, April 1989, 541–580.

[23] Myers, C. J., Rokicki, T. G., Meng, T. H.-Y.: Automatic synthesis of gate-level timed circuits with choice, *Advanced Research in VLSI*, IEEE Computer Society Press, 1995.

[24] Pastor, E.: *Structural Methods for the Synthesis of Asynchronous Circuits from Signal Transition Graphs*, Ph.D. Thesis, Universitat Politècnia de Catalunya, February 1996.

[25] Pastor, E., Cortadella, J.: An efficient unique state coding algorithm for Signal Transition Graphs, *Proc. International Conf. Computer Design (ICCD)*, October 1993.

[26] Pastor, E., Cortadella, J., Kondratyev, A., Roig, O.: Structural Methods for the Synthesis of Speed-Independent Circuits, *IEEE Transactions on Computer-Aided Design*, **17**(11), November 1998, 1108–1129.

[27] Peña, M. A., Cortadella, J.: Combining Process Algebras and Petri Nets for the Specification and Synthesis of Asynchronous Circuits, *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, IEEE Computer Society Press, March 1996.

[28] Rosenblum, L. Y., Yakovlev, A. V.: Signal graphs: from self-timed to timed ones, *Proceedings of International Workshop on Timed Petri Nets*, IEEE Computer Society Press, Torino, Italy, July 1985.

[29] Staradoutsev, N., Bystrov, S., Goncharov, M., Klotchkov, I., Smirnov, A.: Towards Synthesis of Monotonic Asynchronous Circuits from Signal Transition Graphs, *Int. Conf. on Application of Concurrency to System Design*, June 2001.

[30] Tan, S.-Y., Furber, S. B., Yen, W.-F.: The Design of an Asynchronous VHDL Synthesizer, *Proc. Design, Automation and Test in Europe (DATE)*, IEEE Computer Society Press, February 1998.

[31] Vanbekbergen, P.: *Synthesis of Asynchronous Control Circuits from Graph-Theoretic Specifications*, Ph.D. Thesis, Catholic University of Leuven, September 1993.

[32] Varshavsky, V. I., Ed.: *Self-Timed Control of Concurrent Processes: The Design of Aperiodic Logical Circuits in Computers and Discrete Systems*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1990.

[33] Ykman-Couvreur, C., Lin, B., Goossens, G., Man, H. D.: Synthesis and optimization of asynchronous controllers based on extended lock graph theory, *Proc. European Conference on Design Automation (EDAC)*, IEEE Computer Society Press, February 1993.