State Encoding of Large Asynchronous Controllers

Josep Carmona
Universitat Politècnica de Catalunya
Barcelona, Spain

Jordi Cortadella
Universitat Politècnica de Catalunya
Barcelona, Spain

ABSTRACT

A novel method to solve the state encoding problem in Signal Transition Graphs is presented. It is based on the structural theory of Petri nets and can be applied to large specifications with hundreds of signals. This new method opens the door to incorporate logic synthesis in the design flow of large control circuits obtained from high-level specifications. The experimental results validate the quality of the encoded circuits and show the significant improvements that can be obtained by the synthesis of large controllers.

Categories and Subject Descriptors: B.6.3 [Hardware]: Logic Design - Design Aids; J.6 [Computer Applications]: Computer-aided engineering.

General Terms: Algorithms, Design.

Keywords: Asynchronous circuits, Petri nets, state encoding.

1. INTRODUCTION

The state encoding problem in asynchronous circuits can be stated as follows: given a specification describing the interaction between the input and output signals, find an encoding for every state such that the circuit can respond deterministically to the input stimuli as defined by the specification.

Several authors have tackled this problem using different specification formalisms. In the area of asynchronous controllers, the methods proposed for Burst-mode (BM) machines [8] and Signal Transition Graphs (STGs) [4], among others, are the ones that have had the most relevant impact.

State encoding involves different difficult problems. The first one is the *detection of state encoding conflicts*. It can be solved by an exhaustive enumeration of the state space. However, given the potential high degree of concurrency of asynchronous specifications, the state space can be exponential in the size of the specification. Some tools use symbolic techniques (BDDs) to enumerate the states efficiently [5]. However, these techniques are still insufficient for specifications with a large number of signals.

The second problem is the *insertion of new state signals* that disambiguate the encoding conflicts. The insertion must be done in such a way that the new signals are consistent (rising and falling transitions alternate) and their behavior is hazard-free.

This paper faces the problem of state encoding of large asynchronous controllers using STGs. STGs can capture the main paradigms of asynchronous behavior: concurrency, causality and choice.

The only methods that have tackled state encoding for large STGs are restricted to certain subclasses of Petri nets: marked graphs [11] or free-choice Petri nets [13]. This paper proposes a novel method to solve the state encoding problem that is not restricted to any subclass of Petri nets. The complexity is determined by the size of the specification and not by the size of the state space.

One of the main interests of the proposed approach is the capability of handling large STGs generated from HDLs, such as Tangram [1] or Balsa [7]. The main advantage of these HDLs is that they are supported by a complete synthesis flow using the so-called syntax-directed translation paradigm. The state encoding problem is avoided by implicitly over-encoding the system with a netlist of handshake components that implement the parse tree of the specification. However, syntax-directed translation does not benefit from the power of logic synthesis manipulating the Boolean equations of the next-state functions, thus leading to unoptimized implementations.

In [3], 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 synthesis was BM machines, that impose limitations on modeling the inherent concurrency of asynchronous systems and require to handle double-sided timing constraints not always easy to meet.

2. EXAMPLE

The contributions presented in the paper are briefly summarized by solving the encoding problem of a simple VME bus controller. It consists of three entities (the bus, the device and the controller) that interact through a bidirectional buffer according to a given protocol. The protocol can be formally specified with an STG, as shown in Fig. 1(a). STGs are formally defined in Section 3.2.

The underlying Petri net of the STG does not belong to any of the classes for which there are structural methods

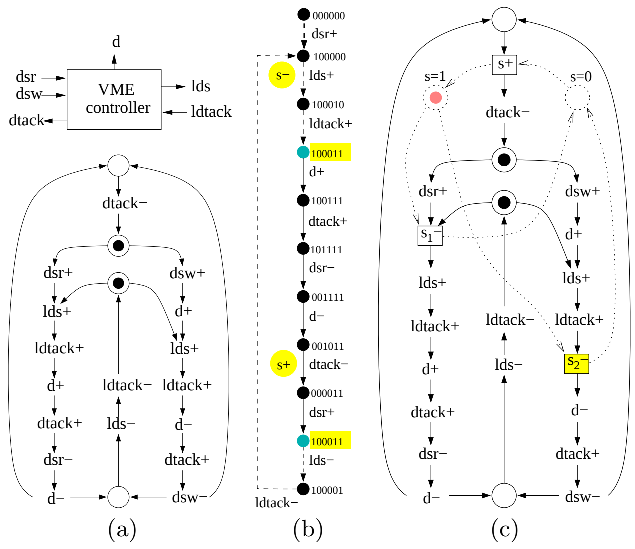


Figure 1: (a) VME Bus controller, (b) Trace connecting a CSC conflict, (c) Signal insertion.

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 one that solves the encoding problem directly on the structure of the Petri net.

The easy part of the state encoding problem

Detecting encoding conflicts in large STGs can be done by using existing techniques based on Petri net unfoldings [10] or integer-linear programming (ILP) models [2]. These methods would reveal the conflict shown in Fig. 1(b), between the states with the shadowed encoding 100011, corresponding to the vector $(dsr, dsw, dtack, d, lds, ldtack)$. A trace connecting the conflicting states would also be reported by the aforementioned methods.

To solve the conflict, a new signal s can be inserted. Two possible insertion points for s^+ and s^- are depicted in Fig. 1(b). These points can be easily identified with their corresponding events in the STG (e.g. $dtack^-$ for s^+ and lds^+ for s^-). Figure 1(c) depicts how the event s^+ has been inserted before $dtack^-$ and s_1^- before lds^+ (we ask the reader to ignore the event s_2^+ and the dotted places and arcs, for the moment). This insertion disambiguates the conflict depicted in Fig. 1(b). In general, other conflicts can also be solved as a by-product of any signal insertion.

However, any signal must have a consistent encoding. Intuitively, consistency means that the rising and falling events of any signal must alternate in any trace of the behavior. 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.

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

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

The difficult part of the state encoding problem

How to insert s^+ and s^- events in such a way that, besides solving the targetted 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. 1(c). However, Petri nets can have very intricate causality, concurrency 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 to *automatically* insert all the events of a new signal in such a way that consistency is guaranteed. The method is based on the structural theory of Petri nets.

We next give a rough idea on how it works. 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. 1(c), they are called $\langle s = 0\rangle$ and $\langle s = 1\rangle$ for signal s, and they are represented as dotted places. These places only interact with events of signal s with arcs $\langle s = 0\rangle \rightarrow s^+ \rightarrow \langle s = 1\rangle$ and $\langle s = 1\rangle \rightarrow s^- \rightarrow \langle s = 0\rangle$.

On the other hand, a place is called *implicit* (or redundant) if it can be removed from the Petri net without changing its behavior. The reader can immediately realize that the behavior of the Petri net of Fig. 1(c) is the same regardless the presence of the dotted places. Therefore, the dotted places are implicit. The proposed method for signal insertion is based on a result from [9]: a signal is consistently encoded if the associated encoding places are implicit.

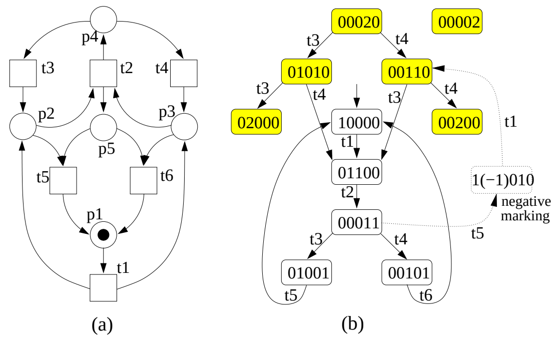
This result is crucial for the method presented in this paper. The reader can realize that the dotted places would not be implicit if the dotted arcs $\langle s = 1\rangle \rightarrow s_2^- \rightarrow \langle s = 0\rangle$ would not be present in the STG with the dotted places.

The result from [9] enables not only to check whether a signal is consistently encoded, but also to *automatically* find the insertion points that guarantee consistency. The problem can be solved using mixed ILP (MILP) models that work directly on the structure of the Petri net and avoid the explicit enumeration of the state graph. Most of the paper will be devoted to formalize the method to automatically insert state signals consistently in an STG.

3. BASIC PETRI NET THEORY

A *Petri Net* (PN) [12] is a 4-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 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 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. An example of Petri net is shown in Fig. 2(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', 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 $[m_0\rangle$, and form a graph called *reachability graph*. A PN is *k-bounded*



$$
\begin{array}{c}
m \\
\left[\begin{array}{c} 0\\0\\1\\1\\0 \end{array}\right]
\end{array}
=
\begin{array}{c}
m_0 \\
\left[\begin{array}{c} 1\\0\\0\\0\\0 \end{array}\right]
\end{array}
+
\begin{array}{c}
N \\
\left[\begin{array}{cccccc}
-1 & 0 & 0 & 0 & +1 & +1\\
+1 & -1 & +1 & 0 & -1 & 0\\
+1 & -1 & 0 & +1 & 0 & -1\\
0 & +1 & -1 & -1 & 0 & 0\\
0 & +1 & 0 & 0 & -1 & -1
\end{array}\right]
\end{array}
\begin{array}{c}
\vec{\sigma} \\
\left[\begin{array}{c} 2\\1\\0\\0\\1\\0 \end{array}\right]
\end{array}
$$

(c)

Figure 2: (a) Petri net, (b) Potential reachability graph, (c) Marking equation.

if no marking in $[m_0\rangle$ assigns more than k tokens to any place. A place in a Petri net is *implicit* if its removal does not affect the behavior of the system.

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

3.1 Petri nets 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 $\#(\sigma, t)$ the number of times that a transition t occurs in σ, we can write the *marking equation* for p as:

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

The marking equations for all the places in the net can be written in the following matrix form (see Fig. 2(c)):

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

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

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

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 satisfying (1) which are not reachable. Those markings are said to be *spurious* [14]. Figure 2(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 is shown in Fig. 2(c). However, m is not reachable by any feasible sequence. Figure 2(b) depicts the graph containing 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 (10000). The marking (00110) is only reachable from the initial state by visiting a negative marking through the sequence $t_1t_2t_5t_1$, as shown in Fig. 2(b). Therefore, equation (1) provides only a sufficient condition for reachability of a marking. For certain subclasses of PNs, e.g. when the net is *free-choice* [12], *live*, bounded and *reversible*, equation (1) together with the set of *traps* of the system completely characterizes reachability [6].

A 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 non existence of a solution for m, σ and s to the following LP model [14], where $P' = P \setminus \{p\}$:

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

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 (necessary condition). The second constraint indicates that all predecessor places, except p, of the transition selected by s have enough tokens to enable it[1]. The third constraint indicates that p has not 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 non existence of a solution guarantees implicitness. The fact that the condition is only sufficient comes from the fact that the marking m could be spurious.

The previous LP model can be slightly transformed and converted to the dual problem (see the details in [14]). Finally, the following result can be derived:

THEOREM 3.1 ([14]). *If $m_0(p)$ is greater than or equal to the optimal value of the following LP problem, then p is implicit:*

$$\begin{array}{rcl}
\min \;\; \mathbf{y} \cdot m_0[P'] + \mu & & \\
\mathbf{y} \cdot \mathbf{N}[P', T] & \leq & \mathbf{N}[p, T] \\
\mathbf{y} \cdot \mathcal{F}[P', p^\bullet] + \mu \cdot \mathbb{1} & \geq & \mathcal{F}[p, p^\bullet] \\
\mathbf{y} & \geq & \mathbb{0}
\end{array}$$

3.2 Signal Transition Graphs

A Signal Transition Graph (STG) is a triple $\langle N, \Sigma, \Lambda \rangle$, where N is a Petri net, Σ is a set of signals (input, output and internal), and Λ is the labeling function $\Lambda : \mathcal{T} \rightarrow \Sigma \times \{+,-\}$, where transitions are interpreted as signal changes. Transitions of a signal $a \in \Sigma$ are denoted by a^+ and a^-, while a^* denotes a generic rising or falling

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

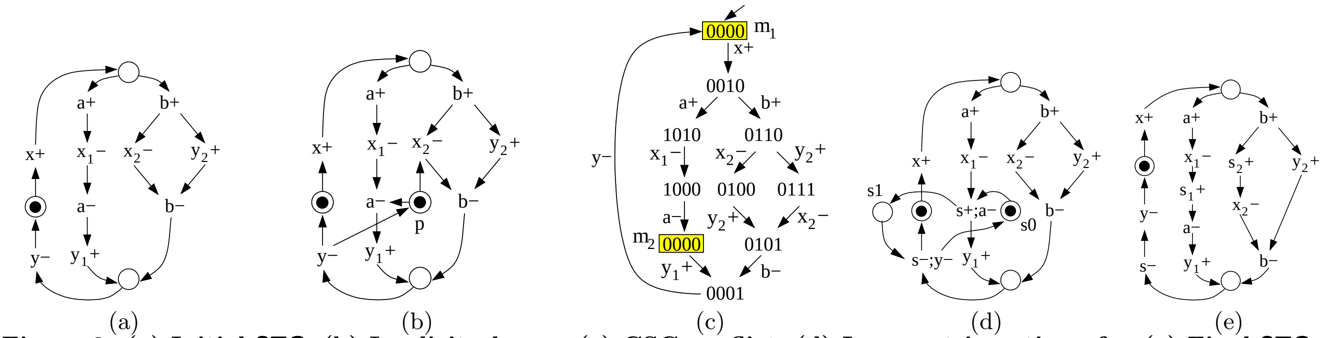


Figure 3: (a) Initial STG, (b) Implicit place p, (c) CSC conflict, (d) Incorrect insertion of s, (e) Final STG.

transition[2]. 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.

Each marking is encoded with a *binary vector* of signal values by means of a labeling function $\lambda : [m_0\rangle \to \{0,1\}^{|\Sigma|}$. The notation $\lambda_s(m)$ will be used to denote the value of signal s at marking m. All markings must be *consistently 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 condition holds, the events that the circuit must produce at each reachable state are uniquely determined by the binary code of the state.

Four conditions are required for an STG to be implementable under the speed-independent delay model: consistency, CSC, *output-persistency* and *boundedness* [5].

4. SOLVING CSC CONFLICTS

The method presented on this paper relies on the following theorem, which bridges consistency and place implicitness:

THEOREM 4.1 ([9]). *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.*

Let us describe the approach with an example. The initial STG is shown in Fig. 3(a), and its state graph in Fig. 3(c). The conflicting states, m_1 and m_2, are encoded with label 0000. The sequences connecting 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^- event 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 decide to break the sequences σ_1 and σ_2 by the transitions a^- and y^-, respectively. The

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

```
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 $s_0 \to t \to s_1$ do
      Relabel the transition as $(t; s^+)$ or $(s^+; t)$
  • for every transition $t$ with arcs $s_1 \to t \to s_0$ do
      Relabel the transition as $(t; s^-)$ or $(s^-; t)$
  • Expand (series or parallel) the relabeled transitions
```

Figure 4: Main algorithm for solving CSC conflicts.

places associated to the new signal s will have arcs connected to these transitions, as shown in Fig. 3(d). However, these new events are not sufficient for s to be consistent: there is an infinite sequence, visiting the event b^+, where only s^- events appear for signal s.

This inconsistency is related to the non-implicitness of s_0. Notice that s_0 is connected to the same transitions to the implicit place p in Fig. 3(b) but x_2^-. The consistency of s can be achieved by adding the arc $s_0 \to x_2^-$ and the complementary arc $x_2^- \to s_1$. This insertion point must be labeled with an occurrence of s^+.

After adding the new events, the insertion points become transitions labeled with a pair of events. They can be expanded by well-known Petri net rules (series or parallel expansion) that preserve all the necessary properties for synthesis [12]. The final STG is shown in Figure 3(e), where places s_0 and s_1 have been removed because they are implicit. The final STG has CSC.

4.1 Main algorithm

The main algorithm for solving CSC is presented in Fig. 4. Finding the conflicts and the corresponding traces can be done by using existing methods to check CSC [2, 10]. 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 that case the new s^* event is inserted as a successor event[3]. In this paper, we only deal with sequential insertion of events. Insertions with more

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

concurrency are also possible, but not discussed due to the lack of space.

4.2 MILP model to find insertion points

The method to insert a new state signal is based on the LP model of theorem 3.1. Instead of checking implicitness, we aim at finding two implicit and complementary places that mimic the value of a new state signal. To achieve this, the original LP model from theorem 3.1 is transformed into a new MILP model as follows:

- A new row $\mathbf{N}[s_0, T]$ of $|T|$ variables, with $\mathbf{N}(s_0, t) \in \{-1, 0, 1\}$, is added to represent the connections of the implicit place s_0. There is no need to add a row for s_1 since it is complementary to s_0, i.e. $\mathbf{N}[s_0, T] = -\mathbf{N}[s_1, T]$.
- The new constraints
$$\begin{array}{rcl} \#(\sigma_1, \mathbf{N}[s_1, T]) & = & \#(\sigma_1, \mathbf{N}[s_0, T]) + 1 \\ \#(\sigma_2, \mathbf{N}[s_0, T]) & = & \#(\sigma_2, \mathbf{N}[s_1, T]) + 1 \end{array}$$
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 $-\mathbf{N}[s_0, T]$.
- The initial marking must be safe: $0 \leq m_0[s_0] \leq 1$.

If the modified MILP model is feasible, the variables $\mathbf{N}[s_0, T]$ represent the necessary arcs to make s_0 and s_1 implicit. For the example of Fig. 3(a), the MILP solver finds the solution

$$\begin{array}{rcccccccccc} & \multicolumn{4}{c}{\overbrace{\hphantom{x^+ \quad a^+ \quad x_1^- \quad a^-}}^{\sigma_1}} & \multicolumn{2}{c}{\overbrace{\hphantom{y_1^+ \quad y^-}}^{\sigma_2}} & & & & \\ & x^+ & a^+ & x_1^- & a^- & y_1^+ & y^- & x_2^- & y_2^+ & b^+ & b^- \\ \mathbf{N}[s_0, T] = [& 0 & 0 & 0 & -1 & 0 & +1 & -1 & 0 & 0 & 0\,] \end{array}$$

The transitions with $\mathbf{N}[s_0, t] = -1$ accommodate the label s^+, whereas the ones with $\mathbf{N}[s_0, t] = +1$ the label s^-. It is important to realize that the transition x_2^-, not belonging neither to σ_1 nor to 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. 3(e) is obtained.

The cost function of the model has another role in our setting: it is designed to choose good insertion points. This has an 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 towards optimizing the following aspects: (1) minimize the number of events of signal s, (2) maximize the number of CSC conflicts solved in σ_1 and σ_2 (other conflicts may exist besides the ones with m_1 and m_2) and (3) increase the number of lock relations [15] with the other signals in σ_1 and σ_2. The details of these optimizations are omitted due to the lack of space.

The method can be easily extended to non-reversible systems by ignoring the trace σ_2 in the MILP model. This trace is not strictly necessary for the correct insertion of signals, however it helps to find better insertion points when using the cost function for optimization.

4.3 Incompleteness of the method

Even though the method presented in this paper is applicable to general Petri nets, it cannot guarantee a solution even in the case it exists. An example of this situation is depicted in Fig. 5 (a and b are inputs, x and y are outputs).

The STGs in Fig. 5(a) and 5(b) have the same behavior. However, the one in Fig. 5(a) is more compact, since the

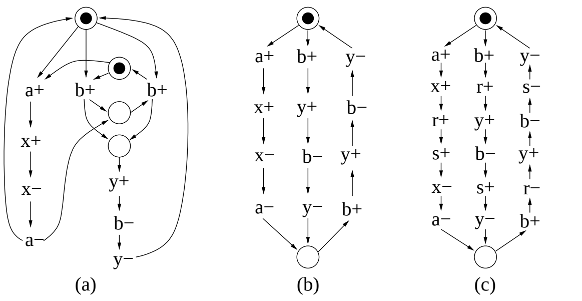


Figure 5: Example with unsolvable CSC conflict.

transitions $y^+b^-y^-$ represent two different subsequences of the behavior. A state-based tool, such as petrify [5], can easily find a solution for (a), like the one in Fig. 5(c), with the insertion of the signals r and s. However, this solution requires to restructure the net (a) by unfolding the shared subsequences and enable a different signal insertion for each one. 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 (b) but not for (a).

Characterizing the class of STGs for which a solution can be guaranteed is a topic under investigation. We believe that well-structured STGs, typically obtained from the synthesis of high-level specifications, belong to this class.

5. EXPERIMENTAL RESULTS

The goal of the experimental results presented in this section is threefold: (a) evaluate the quality of the method by comparing the results with a state-based method (e.g. [4]), (b) evaluate the capability for handling large specifications and (c) show the potential benefits of logic synthesis methods with regard to syntax-directed translation.

Table 1 evaluates the quality of the results when compared to petrify [5]. The examples from [4] have been used[4]. The number of literals of the Boolean equations (in factored form) and the number of state signals inserted to solve CSC are reported[5]. In general, the complexity of the circuits is similar. However, the new method inserts more state signals. The reason for that is because petrify attempts to maximize the number of conflicts solved for each signal, whereas the new approach tries to reduced the number of events of each new signal. In 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.

State-based methods cannot handle large STGs like the ones in Table 2. The benchmarks ART(n, m) model n pipelines of length m synchronized only on its starting points, thus exhibiting a high degree of concurrency. Benchmarks PPWK and PPARB are another type of pipelines, described in [10]. VAR(n, m) models the handshakes of a set of n read- and m write-processes into a 1-bit variable.

The rest of examples are typical netlists of handshake

[4]Only 4 examples with irreducible conflicts have been omitted, since the conflicts cannot be solved unless timing assumptions on the environment are used.

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

example	petrify	MILP	example	petrify	MILP
adfast	14/2	17/2	tsend-bm	39/2	40/3
ircv-bm	38/2	46/4	alloc-outbound	16/2	16/2
mmu	29/3	27/3	duplicator	19/2	16/3
mmu0	29/3	33/5	mod4_counter	25/2	28/4
mmu1	32/3	25/2	ram-read-sbuf	18/1	19/1
mr0	45/3	34/4	sbuf-ram-write	22/2	29/2
mr1	35/4	29/4	sbuf-read-ctl	15/1	15/1
nak-pa	18/1	18/1	master_1882	38/1	38/1
nowick	14/1	13/1	trcv-bm	36/2	41/4
par4	32/4	32/4	seq_mix	20/3	20/2
seq8	47/4	43/6	spec_seq4	20/3	20/3
total				601/52	599/62

Table 1: Results (literals/state signals).

components from an HDL like Tangram [1] or Balsa [7]: an n-way parallelizer (Par(n)), a combination of sequencers and parallelizers (SeqPar(s, p)), and a combination of sequencers, parallelizers and mixers (SPM(s, p, m)). The specifications have been obtained by hiding all the internal channels of the netlists and keeping only the events of the external signals.

The columns report the number of places, transitions and input/output signals of the STG, respectively. For synthesis, structural methods based on projections [2] have been used. The CPU time (in minutes) for state encoding and logic synthesis is reported (enc/syn). None of the previously existing techniques has been able to solve the encoding problem with examples of such size. It is remarkable to realize that the tool has been able to solve one of the examples (Art(20,9)) by inserting 57 signals. Moreover, the power of logic synthesis can be seen in the last three examples, where the literals of the equivalent Tangram implementations (column HDL) are also reported[6]. The results show drastic improvements, e.g. up to 63% in number in literals for SPM(7,16,18).

Finally, Table 3 reports the synthesis results for an n-way sequencer ($n \in [2, 9]$). The results show the superiority of the MILP-based method with respect to petrify and the significant improvements obtained with regard to syntax-directed translation when the size of the circuit grows.

6. CONCLUSIONS

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 towards taking advantage of the optimizations that logic synthesis can do in the Boolean domain. These optimizations cannot be applied unless the state encoding problem is solved for real-life specifications.

Acknowledgment

The authors would like to thank Jose Manuel Colom for his suggestions and helpful discussions. This work has been supported by CICYT TIN2004-07925 and a Distinction for Research from the Generalitat de Catalunya.

7. REFERENCES

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

[2] J. Carmona and J. Cortadella. ILP models for the synthesis of asynchronous control circuits. In *Proc. International Conf. Computer-Aided Design (ICCAD)*, pages 818–826, Nov. 2003.

[3] T. Chelcea, A. Bardsley, D. Edwards, and S. M. Nowick. A burst-mode oriented back-end for the Balsa synthesis system. In *Proc. Design, Automation and Test in Europe (DATE)*, pages 330–337, Mar. 2002.

[4] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. A region-based theory for state assignment in speed-independent circuits. *IEEE Transactions on Computer-Aided Design*, 16(8):793–812, Aug. 1997.

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

[6] J. Desel and J. Esparza. Reachability in cyclic extended free-choice systems. *TCS 114, Elsevier Science Publishers B.V.*, 1993.

[7] D. Edwards and A. Bardsley. Balsa: An asynchronous hardware synthesis language. *The Computer Journal*, 45(1):12–18, 2002.

[8] R. M. Fuhrer, B. Lin, and S. M. Nowick. Symbolic hazard-free minimization and encoding of asynchronous finite state machines. In *Proc. International Conf. Computer-Aided Design (ICCAD).* IEEE Computer Society Press, 1995.

[9] F. García-Vallés and J. M. Colom. Structural analysis of signal transition graphs. In D. H. I. B. Farwer and M. Stehr, editors, *Proceedings of the Workshop Petri Nets in System Engineering (PNSE'97). Modelling, Verification and Validation*, pages 123–134, 1997.

[10] V. Khomenko, M. Koutny, and A. Yakovlev. Detecting state coding conflicts in STG unfoldings using SAT. In *Int. Conf. on Application of Concurrency to System Design*, June 2003.

[11] 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. European Design and Test Conference*, pages 178–183. IEEE Computer Society Press, 1994.

[12] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–574, Apr. 1989.

[13] E. Pastor and J. Cortadella. An efficient unique state coding algorithm for signal transition graphs. In *Proc. International Conf. Computer Design (ICCD)*, pages 174–177, Oct. 1993.

[14] M. Silva, E. Teruel, and J. M. Colom. Linear algebraic and linear programming techniques for the analysis of place/transition net systems. In Reisig, W. and Rozenberg, G., editors, *Lecture Notes in Computer Science: Lectures on Petri Nets I: Basic Models*, volume 1491, pages 309–373. Springer-Verlag, 1998.

[15] P. Vanbekbergen. *Synthesis of Asynchronous Control Circuits from Graph-Theoretic Specifications.* PhD thesis, Catholic University of Leuven, Sept. 1993.

[6]The cost of sequencers, parallelizers and mixers is 8, 23 and 12, respectively [1]. The cost of a C element is considered to be 5 literals ($c^+ = ab + c(a + b)$).

benchmark	P/T	in/out	CPU(min)	lit/sig	HDL
Art(10,9)	216/198	0/99	1.3/2.3	305/28	–
Art(20,9)	436/398	0/199	28.3/44.7	629/57	–
PpWk(3,12)	142/74	0/37	0.5/0.5	190/3	–
PpArb(3,12)	164/90	3/40	10/1.5	206/2	–
Var(9,5)	302/338	73/77	4.7/1.7	613/24	–
Var(12,1)	368/394	86/97	10.8/1.9	445/27	–
Par(12)	63/52	26/26	0.1/0.1	101/12	253
SeqPar(21,10)	160/128	32/32	0.5/1.7	269/23	398
SPM(7,16,18)	192/144	30/30	8.3/3.5	237/17	640

Table 2: State encoding of large controllers.

n	2	3	4	5	6	7	8	9
tangram	8	16	24	32	40	48	56	64
petrify	8	14	20	30	34	39	48	52
MILP	8	14	20	26	31	34	40	44

Table 3: Synthesis of an n-way sequencer (literals).