Methodology and tools for state encoding in asynchronous circuit synthesis *

Jordi Cortadella, Univ. Politecnica de Catalunya, Barcelona, Spain
Michael Kishinevsky, Alex Kondratyev, The University of Aizu, Japan
Luciano Lavagno, Politecnico di Torino, Italy
Alex Yakovlev, University of Newcastle upon Tyne, United Kingdom

Abstract

This paper proposes a state encoding method for asynchronous circuits based on the theory of *regions*. A region in a Transition System is a set of states that "behave uniformly" with respect to a given transition (value change of an observable signal), and is analogue to a place in a Petri net. Regions are tightly connected with a set of *properties* that must be preserved across the state encoding process, namely: (1) trace equivalence between the original and the encoded specification, and (2) implementability as a speed-independent circuit. We build on a theoretical body of work that has shown the significance of regions for such property-preserving transformations, and describe a set of algorithms aimed at efficiently solving the encoding problem. The algorithms have been implemented in a software tool called `petrify`. Unlike many existing tools, `petrify` represents the encoded specification as an STG, and thus allows the designer to be more closely involved in the synthesis process. The efficiency of the method is demonstrated on a number of "difficult" examples.

1 Introduction

In the last decade, Signal Transition Graphs (STGs) [7, 1] have attracted much of the attention of the asynchronous circuit design community due to their inherent ability to capture the main paradigms of asynchronous behaviour: causality, concurrency and data-dependent and non-deterministic choice. STGs are Petri nets whose events are interpreted with signal transitions of a modeled circuit. The STG model, exactly like "classical" Flow Table models, may require some state signals to be added to those initially specified by the designer. Adding those state signals is commonly referred to as solving the Complete State Coding (CSC) problem.

Since [1] a number of different techniques have been proposed to solve the CSC-problem. The first totally general method, described in [8], used an algorithm whose complexity *practically* precluded any optimization, but produced only one, often suboptimal, solution. The most recent method [9] is based on the concept of an *excitation region* for a signal transition (a set of states in which a signal is enabled to change its value). It has been able to improve on [8] by adopting a *coarser granularity* in the exploration of the solution space. This coarser granularity has a price, though: as we will show in Section 6, there is a number of examples of STGs which could not be solved by their method (nor by previous ones, mainly due to the large number of states), unless changes in the specification (e.g., reductions in concurrency) are allowed. Moreover, the authors could not characterize the class of STGs for which their method was guaranteed to find a solution.

Our approach differs from the previous work in the area, because it is based on the notion of *regions of states*, which is more general than, albeit related to, that of *excitation regions* (an excitation region is a *specific intersection* of regions). By exploring a broader design space than [9], we can thus solve a larger number of problems, and potentially reach better solutions especially in terms of *circuit performance*. For example, our approach can *efficiently* trade off logic complexity with execution speed, by changing the level of parallelism with which state signal transitions are inserted. On the other hand, our search space is still reduced with respect to [8], and thus we can claim better control on the quality of the solution.

This paper is organised as follows. Section 2 provides some theoretical background (the interested reader is referred to [2] for the details). Sections 3 and 4 define the idea of property-preserving event insertion and apply it to solving the CSC problem. Sections 5 and 6 describe implementation aspects and experimental results.

2 Theoretical background

2.1 Transition systems and Petri nets

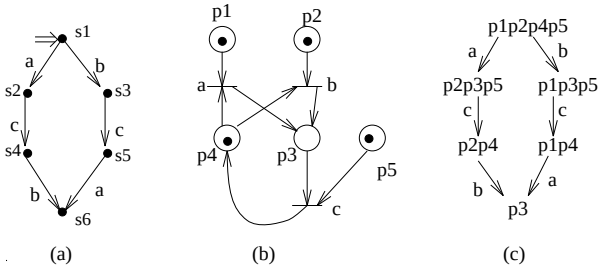

Figure 1: A TS (a), the corresponding PN (b), its RG (c)

Informally, a TS ([6]) can be represented as an arc-labeled directed graph. A simple example of a TS without cycles is shown in Figure 1,a. A TS is called *deterministic* if for each state s and each

* This work has been partially supported by grant CICYT TIC 95-0419 (J. Cortadella), EPSRC visiting fellowship GR/J78334 (M. Kishinevsky), MURST project "VLSI architectures" (L. Lavagno), and EPSRC grant GR/J52327 (A. Yakovlev).

33rd Design Automation Conference ®

DAC 96 - 06/96 Las Vegas, NV, USA

label a there can be at most one state s' such that $s \xrightarrow{a} s'$. A TS is called *commutative* if whenever two actions can be executed from some state in any order, then their execution always leads to the same state, regardless of the order.

A Petri Net is a quadruple $N = (P, T, F, m_0)$, where P is a *finite* set of places, T is a *finite* set of transitions, $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation, and m_0 is the initial marking. A transition $t \in T$ is enabled at marking m_1 if all its input places are marked. An enabled transition t may fire, producing a new marking m_2 with one less token in each input place and one more token in each output place ($m_1 \xrightarrow{t} m_2$). A PN expressing the same behavior as the TS from Figure 1,a is shown in Figure 1,b.

The set of all markings reachable in N from the initial marking m_0 is called its Reachability Set. A net is called *safe* if no more than one token can appear in a place in any reachable marking. The graph with vertices corresponding to markings of a PN and with an arc (m_1, m_2) in the graph if and only if $m_1 \rightarrow m_2$ is called its Reachability Graph (RG). One can easily check that the RG Figure 1,c derived for the PN from Figure 1,b is isomorphic to the TS (Figure 1,a).

2.2 Regions and Excitation Regions

Let S_1 be a subset of the states of a TS, $S_1 \subseteq S$. If $s \notin S_1$ and $s' \in S_1$, then we say that transition $s \xrightarrow{a} s'$ *enters* S_1. If $s \in S_1$ and $s' \notin S_1$, then transition $s \xrightarrow{a} s'$ *exits* S_1. Otherwise, transition $s \xrightarrow{a} s'$ *does not cross* S_1. A *region* is a subset of states with which *all* transitions labeled with the same event e have exactly the same "entry/exit" relation. This relation will become the predecessor/successor relation in the Petri net.

Let us consider the TS shown in Figure 1. The set of states $r_3 = \{s_2, s_3, s_6\}$ is a region, since all transitions labeled with a and with b enter r_3, and all transitions labeled with c exit r_3. On the other hand, $\{s_2, s_3\}$ is not a region since transition $s_1 \xrightarrow{b} s_3$ enters this set, while another transition also labeled with b, $s_4 \xrightarrow{b} s_6$, does not.

A region r is a *pre-region* of event e if there is a transition labeled with e which exits r. A region r is a *post-region* of event e if there is a transition labeled with e which enters r. The set of all pre-regions and post-regions of e is denoted with ${}^\circ e$ and e° respectively.

While regions in a TS are related to places in the corresponding PN, an excitation region for event a is a maximal set of states in which transition a is enabled. Therefore, excitation regions are related to transitions of the PN. A set of states is called an *excitation region* for event a (denoted by $ER_j(a)$) if it is a *maximal connected* set of states such that for every state $s \in ER_j(a)$ there is a transition $s \xrightarrow{a}$. Since any event a can have several separated ERs, an index j is used for the distinction between *different connected occurrences* of a in the TS. In the TS from Figure 1,a there are two excitation regions for event a: $ER_1(a) = \{s_1\}$ and $ER_2(a) = \{s_5\}$. Similarly to ERs, we define switching regions as connected sets of states reached *immediately after* the occurrence of an event.

3 Property-preserving event insertion

Event insertion is informally seen as an operation on a TS which selects a subset of states, splits each state in it into two states and creates, on the basis of these new states, an excitation and switching region for a new event. Figure 2 shows the chosen insertion scheme, analogous to that used by most authors in the area, in the three main cases of insertion with respect to the position of the states in the insertion set $ER(x)$ (*entrance* to, *exit* from or *inside* $ER(x)$).

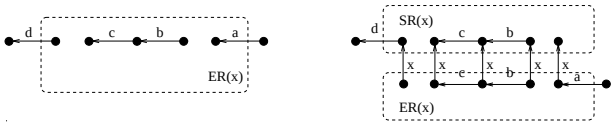

Figure 2: Event insertion scheme

State signal insertion must also preserve the *speed-independence* of the original specification, that is required for the existence of a hazard-free asynchronous circuit implementation.

An event a of a TS A is said to be *persistent in a subset* S' *of states* of S iff $\forall s1 \in S', b \in E : [s1 \xrightarrow{a} \wedge (s1 \xrightarrow{b} s2) \in T] \Rightarrow s2 \xrightarrow{a}$. An event is said to *persistent* if it is persistent in S. For a binary encoded TS, determinism, commutativity and output event persistency guarantee speed-independence of its circuit implementation. Formally, we say that an insertion state set $ER(x)$, in a TS A' obtained from a deterministic and commutative TS A by inserting event x, is a *speed-independence preserving subset (SIP-set)* iff: (1) for each $a \in E$, if a is persistent in A, then it remains persistent in A', and (2) A' is deterministic and commutative.

The following two properties of insertion sets, based on theory developed in [2], link together the notions of TS *regions* and *SIP-sets* and provide a rationale for our approach.

Property 3.1

- *(P1) If r is a region in a commutative and deterministic* TS*, then r is an SIP-set.*
- *(P2) If r is an excitation region of an event a in a commutative and deterministic* TS *and a is persistent in r, then r is an SIP-set.*
- *(P3) If r_1 and r_2 are pre-regions of the same event in a commutative and deterministic* TS*, $r_1 \cap r_2$ is connected and all exit events of $r_1 \cap r_2$ are persistent, then $r_1 \cap r_2$ is a SIP-set.*

These properties suggest that the good candidates for insertion sets should be sought on the basis of regions and their intersections (while the approach of [9] could exploit only case P2). Since any disjoint union of regions is also a region, this gives an important corollary that nice sets of states can be built very efficiently, from "bricks" (regions) rather than "sand" (states).

4 Solving Complete State Coding

A Signal Transition Graph (STG, [1, 7]) is a Petri net labeled with up and down transitions of a set of signals (denoted by x^+ and x^- for signal x respectively).

A necessary condition for STG implementability is *consistent labeling*. Informally, this means that in every firing sequence from the initial marking, rising and falling transitions alternate for each signal. In other words, each marking can be uniquely labeled with a vector of signal values. Once consistency is ensured, *Complete State Coding* (CSC) becomes necessary and sufficient for the existence of a logic circuit implementation. A consistent STG satisfies the CSC property if for every pair of states s, s' of the associated TS, such that $v(s) = v(s')$, the set of non-input transitions enabled in both is the same.

Assume that the set of states S in a TS is partitioned into two subsets which are to be encoded by means of an additional signal to solve some CSC conflicts. Let r and $\overline{r} = S - r$ denote the blocks of such a partition. In order to implement such an encoding, we need to insert appropriate transitions of the new signals in the *border states* between the two subsets.

In this paper we shall consider the so-called *exit border* (EB) of a partition block r, denoted by $EB(r)$, which is informally a subset

of states of r with transitions exiting r. We call $EB(r)$ *well-formed* if there are no transitions leading from states in $EB(r)$ to states in $r - EB(r)$.

Consider the example in Figure 3 (enabled signals have their value followed by $*$ in the signal label). State pair $(1^*1,\ 1^*1^*)$ has a CSC conflict, assuming that signal a is input and b is non-input, and so do $(1^*1,\ 1^*1^*)$ and $(0^*1, 01^*)$ (while $(00^*, 0^*0^*)$ does not, because b is enabled in both). The partition $r = r2,\ \overline{r} = r2'$ separates all conflicting pairs, and can thus be tentatively used to solve the conflicts. The borders, in this case, are denoted by the shaded areas. If they are selected as excitation regions for the new signal y, we obtain the TS (c). Note that some border states are conflicting. This means that the new TS will still have *secondary* CSC problems, that must be solved by iterating the procedure (the proof of convergence is given in [2]).

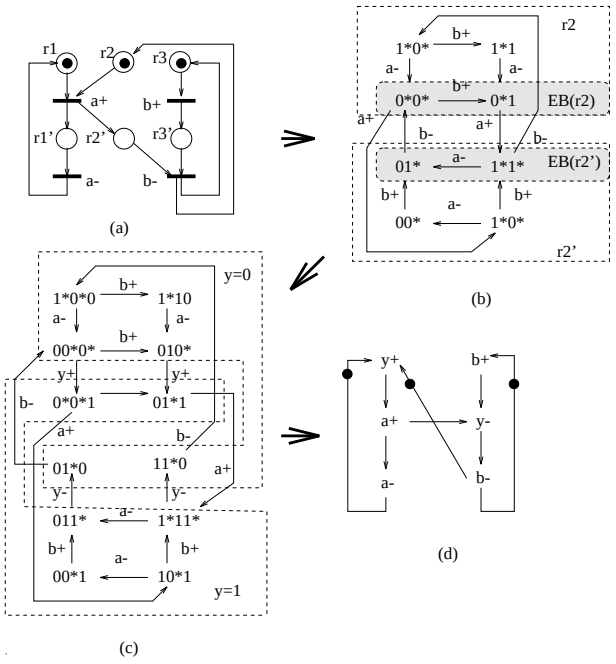

Figure 3: Illustration of event insertion

Note that we need each new signal x to orderly cycle through states in which it has value 0, 0^*, 1 and 1^*. We can formalize this requirement with the notion of *I-partition* ([8] used a similar definition).

Given a TS $TS = (S, T, E, s_{in})$, an I-partition is a partition of S into four blocks: S^0, S^1, S^+ and S^-. $S^0(S^1)$ defines the states in which x will have the value 0 (1). $S^+(S^-)$ defines ER$(x+)$ (ER$(x-)$). For a consistent encoding of x, the only allowed events crossing boundaries of the blocks are the following: $S^0 \rightarrow S^+ \rightarrow S^1 \rightarrow S^- \rightarrow S^0$, $S^+ \rightarrow S^-$ and $S^- \rightarrow S^+$ (the latter two would cause a persistency violation, though).

The problem of finding an I-partition is reduced to finding a bipartition S. Each block b of S induces a bipartition $\{b, \overline{b}\}$, $(\overline{b} = S \backslash b)$. Given a block b, an I-partition can be calculated by defining S^+ and S^- with the following recursion:

1. $\{s \in b \mid \exists\, s \rightarrow s' \ \wedge\ s' \in \overline{b}\} \subseteq S^+$
   $\{s \in \overline{b} \mid \exists\, s \rightarrow s' \ \wedge\ s' \in b\} \subseteq S^-$
2. $[s \in S^+ \ \wedge\ s' \in b \ \wedge\ s \rightarrow s'] \ \Rightarrow\ s' \in S^+$
   $[s \in S^- \ \wedge\ s' \in \overline{b} \ \wedge\ s \rightarrow s'] \ \Rightarrow\ s' \in S^-$

and finally $S^0 = b - S^+$ and $S^1 = \overline{b} - S^-$. The sets of states defined by condition 1 correspond to the smallest "legal" exit border of b with respect to $\overline{b}$ $(EB(b))$. The additional states of condition 2 define the smallest well-formed EBs. We will denote by MWFEB(b) the *minimal well-formed* EB of b.

The set of candidates explored by our encoding algorithm will be restricted to be an I-partition by construction. We proved in [2] that the method is complete, in that it can solve CSC for any safe, consistent, output-persistent STG.

```
bricks = calculate_all_bricks ()
frontier = good_blocks = {the best FW bricks}
repeat /* heuristic search */
      new_frontier = ∅
      for each bl ∈ frontier do
            for each br ∈ bricks adjacent to bl do
                  new_bl = bl ∪ br
                  if cost(new_bl) < cost(bl) then
                        good_blocks = good_blocks ∪ {new_bl}
                        new_frontier = new_frontier ∪ {new_bl}
      frontier = select the best FW blocks from new_frontier
until new_frontier = ∅
return the best block in good_blocks
```

Figure 4: Heuristic search to find a block for event insertion

5 A heuristic-search strategy to solve CSC

The main algorithm for the insertion of one state signal is as follows:

1. Generate a set of I-partitions that preserve speed independence (figure 4)
2. Estimate the cost of the generated I-partitions
3. Select the best I-partition
4. Increase the concurrency of the inserted signal

Initially, all bricks of the TS are calculated by (1) obtaining all minimal regions of the TS and (2) calculating all possible intersections of pre-/post-regions of the same event. Since the number of pre- and post-regions of an event is usually small, an exhaustive generation is feasible.

The best block for event insertion is obtained as the union of adjacent bricks. At each iteration of the search, a frontier of FW (frontier width, a parameter trading off solution quality versus time) "good" blocks is kept. Each block is enlarged by adjacent bricks and the new obtained blocks are considered candidates for the next iteration only if they are "better", according to the cost function, than their ancestors. The final block for insertion is calculated as the union of best disconnected blocks. A greedy block merging approach guided by the cost function is used.

Given a block b, S^+ and S^- are initially calculated as the MWFEB of b and $\overline{b}$ respectively. This leads to a solution with minimum concurrency of the inserted event. Concurrency can be increased by enlarging S^+ and/or S^-([8]). In our approach, after having calculated the best configuration for event insertion, S^+ and S^- are greedily enlarged by adding bricks that are adjacent to them. The enlargement is only accepted if the new configuration improves the cost of the solution. The following factors are considered in the cost function for the insertion of signal x (in order of priority):

- ER$(x+)$ and ER$(x-)$ must be SIP blocks.
- The insertion of x must not modify the specification of the environment (e.g., x cannot be inserted before input events).

benchmark	places	trans.	signals	states	CPU
master-read	37	26	18	18856	927
master-read×2	74	52	38	5.4×10^8	10849
par8	43	36	26	1.7×10^6	1175
par16	83	68	40	2.8×10^{12}	13546
pipe8	24	16	19	87480	371
pipe16	48	32	38	1.9×10^9	15689

Table 1: Results for STGs with a large number of states

- The number of solved CSC conflicts must be maximized.
- The estimated complexity of the circuit must be minimized.

In the current implementation, the complexity of the circuit is approximated by the sum of the number of *trigger signals* for each ER. Each trigger signal labels one of the transitions which enter an ER and corresponds to a fan-in signal in the implementation. More accurate estimations are foreseen for future implementations.

6 Experimental results

The region-based approach presented in this paper has been integrated in `petrify`, a tool for the synthesis of Petri nets [3]. We have used several benchmarks that no other automatic tool, such as SIS or ASSASSIN, has been able to solve. Some of them are even difficult to solve manually by expert designers. Our approach has succeeded in all of them.

One of the most important features of the CSC algorithm implemented in `petrify` is the capability of managing extremely large state graphs generated from STGs with high concurrency. Two factors are essential for this capability: (1) the symbolic representation and manipulation of the state graph by means of *Ordered Binary Decision Diagrams* (2) the exploration of blocks of states at the level of regions rather than states. Table 1 presents the CPU times (in seconds on a SPARCSTATION 20) required to satisfy CSC for some examples with a vast state space, which cannot be solved in a reasonable amount of memory or time by SIS or ASSASSIN.

Table 2 reports the results obtained with `petrify` in comparison with the ones obtained by ASSASSIN ([5]). The quality of the results is comparable to those obtained by ASSASSIN. Even with the estimation of logic performed in `petrify`, ASSASSIN can still offer slight improvements in a few examples. This means that an estimation of logic based on only trigger signals is not accurate enough.

7 Conclusions

In this paper we have presented a method and associated algorithms for solving state coding problems by means of state signal insertion. Our main target here was solving Complete State Coding problem, one of fundamental issues in asynchronous circuit synthesis from Signal Transition Graphs. We believe that our approach to: (1) Transition System partitioning, (2) new signal insertion, and (3) reconstruction of the model in Petri net form, based on the concept of region of states, will prove useful in solving other problems in asynchronous circuit synthesis. In particular, the technology mapping problem for Speed-Independent circuits ([4]) can be cast in this form.

		ASSASSIN		petrify	
benchmark	states	area	CPU	area	CPU
adfast	44	390	0.4	294	10.5
nak-pa	56	456	0.7	456	4.8
alloc-outbound	17	350	0.1	350	5.4
nowick	18	340	0.1	428	2.6
ram-read-sbuf	36	406	0.2	406	6.0
sbuf-ram-write	58	764	0.7	300	23.9
sbuf-read-ctl	15	244	0.0	244	1.4
mux2	99	1386	3.0	1774	142.2
postoffice	58	1094	1.0	800	0.0
duplicator	20	294	0.1	294	5.9
spec_seq4	20	236	0.1	236	6.2
seq_mix	20	324	0.1	324	7.5
seq8	36	480	0.4	480	37.8
trcv-bm	44	826	0.6	824	56.5
tsend-bm	41	1010	0.6	962	0.0
ircv-bm	44	842	0.4	1042	64.3
mod4_counter	16	648	0.1	648	0.0
master-read	1882	726	607.7	750	75.7
mmu	174	698	10.6	732	51.9
mr0	302	1008	40.0	626	153.6
mr1	190	912	17.9	650	23.0
mmu0	174	886	8.4	610	48.4
mmu1	82	700	1.8	514	45.0
par_4	628	506	206.4	506	88.0
divider8	18	848	0.4	914	18.7
vme2int	74	1014	0.8	938	44.4
combuf2	11	270	0.2	262	3.7
total		17658	902.8	16364	927.4

Table 2: Experimental results compared with ASSASSIN

References

[1] T.-A. Chu. On the models for designing VLSI asynchronous digital systems. *Integration: the VLSI journal*, 4:99–113, 1986.

[2] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. A region-based theory for state assignment in asynchronous circuits. Technical Report 95-2-006, University of Aizu, Japan, October 1995.

[3] J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev. Synthesizing Petri nets from state-based models. In *Proceedings of the International Conference on Computer-Aided Design*, November 1995.

[4] A. Kondratyev, M. Kishinevsky, B. Lin, P. Vanbekbergen, and A. Yakovlev. Basic gate implementation of speed-independent circuits. In *Proceedings of the Design Automation Conference*, 1994.

[5] Bill Lin, Chantal Ykman-Couvreur, and Peter Vanbekbergen. A general state graph transformation framework for asynchronous synthesis. In *Proceedings of the European Design Automation Conference (EURO-DAC)*, pages 448–453. IEEE Computer Society Press, September 1994.

[6] M. Nielsen, G. Rozenberg, and P.S. Thiagarajan. Elementary transition systems. *Theoretical Computer Science*, 96:3–33, 1992.

[7] L. Y. Rosenblum and A. V. Yakovlev. Signal graphs: from self-timed to timed ones. In *International Workshop on Timed Petri Nets*, 1985.

[8] P. Vanbekbergen, B. Lin, G. Goossens, and H. De Man. A generalized state assignment theory for transformations on Signal Transition Graphs. In *Proceedings of the International Conference on Computer-Aided Design*, pages 112–117, November 1992.

[9] C. Ykman-Couvreur and B. Lin. Optimised state assignment for asynchronous circuit synthesis. In *Proc. Second Working Conf. on Asynchronous Design Methodologies, London*, May 1995.