,. Conclusions and further research

A normal form has been defined for delay-insensitive process expressions. By showing hat every process expression can be transformed to this normal form using the laws 'resented in [7] (most of which have been presented in this paper) we have shown that hese laws completely characterize the semantics of delay-insensitive processes.

The paper uses a general approach towards proving an algebra to be complete. The pproach was taken from Roscoe and Hoare in [8]. Actually, the approach gives clues s to which laws to introduce. The purpose of each law is to make a step towards an xpression's normal form.

The last step in the transformation to normal form turned out to be nontrivial. Espe-ially tricky there is the proof that the transformation procedure terminates.

With the proofs of completeness and soundness of the algebra, the way is clear to use ıe algebra for actual delay-insensitive circuit design. In view of the size of correctness roofs in the algebra a proof editor or proof assistant to show equivalence or refinement clearly called for. However, for reasons of practicality we need to design more efficient gorithms, in order to avoid having to reduce all expressions to normal form.

. Acknowledgments

This research was partially funded by the ESPRIT Basic Research Working Group 225 — ACiD-WG and the ESPRIT project EXACT (6143).

.EFERENCES

H. M. Groenboom, M. B. Josephs and J. T. Udding. *Completeness of a delay-insensitive algebra.* Technical Report CS9108, Groningen University, Dept. of Comp. Sci., 1991.

M. Hennessy. *Algebraic Theory of Processes.* Series in Foundations of Computing. The MIT Press, Cambridge, Mass., 1988.

M. B. Josephs and J. T. Udding. An algebra for delay-insensitive circuits. Report WUCS-89-54, Dept. of Comp. Sci. Washington University, St. Louis, Mo, march 1990.

M. B. Josephs and J. T. Udding. *The design of a delay-insensitive stack.* Technical Report CS9004, Groningen University, Dept. of Comp. Sci., 1990.

M. B. Josephs and J. T. Udding. Delay-insensitive Circuits: an Algebraic Approach to their Design. In J.C.M. Baeten and J.W. Klop, editors, *CONCUR'90 Theories of Concurrency: Unification and Extension*, LNCS 458 pages 342 – 366, Springer-Verlag, 1990.

M. B. Josephs. Receptive Process Theory. To appear in *Acta Informatica, 1992.* Also Technical Report, Eindhoven University, Eindhoven, 1990.

P. G. Lucassen and J. T. Udding. A failures model for delay-insensitive processes. In J. van Leeuwen, editor, *Proceedings of CSN 91*, pp. 405-421, 1991. Also Technical Report CS9107, Groningen University, Dept. of Comp. Sci. 1991.

A. W. Roscoe and C. A. R. Hoare. The laws of occam programming. *Theoretical Computer Science*, 60(2):177-229, 1988.

Asynchronous Design Methodologies (A-28)
S. Furber and M. Edwards (Editors)
Elsevier Science Publishers B.V. (North–Holland)
© 1993 IFIP. All rights reserved.

Synthesis of Asynchronous Control Circuits from Symbolic Signal Transition Graphs

A.V. Yakovlev[a], A.I. Petrov[b] [1] and L.Ya. Rosenblum[c]

[a]Department of Computing Science, University of Newcastle upon Tyne, U.K.

[b]Laboratory of Automation Technology, Helsinki University of Technology, Finland

[c]NID Inc., Cambridge, MA, USA

Abstract

Signal Transition Graphs (STGs), a labelled version of Petri Nets (PNs), have recently become popular as a model for asynchronous control circuit synthesis. They are however unable to define the behaviour of abstract Asynchronous Control Structures (ACSs), whose components may have multiple outputs and "signals" are multi-valued or symbolic. We introduce a model, called Symbolic Signal Transition Graph (SSTG), that is a natural extension of STG in which PN transitions are labelled with the changes of values of symbolic variables. We present a synthesis procedure and sufficient conditions for the implementability of the binary expansion (after an appropriate encoding) of an SSTG specification of an abstract ACS. Two circuit synthesis examples, a bus interface and a two-way pipeline channel, illustrate the approach.

Keyword Codes: B.6.1; B.4.2
Keywords: Design Styles; Input/Output Devices

1. INTRODUCTION

As more attention is drawn to system-level VLSI design asynchronous circuits are becoming a crucial part of intercomponent interfaces and various control-dominated structures. The two major tasks in the circuit design are as follows: (i) specifying in a clear, preferably sufficiently abstract manner, the behaviour of the system; and (ii) implementing this behaviour correctly by an interconnection of primitive components. A number of formal specification languages and synthesis methodologies have been proposed in [10, 3, 7, 4, 1, 5]. Amongst them, the Signal Transition Graph (STG) model has acquired probably highest interest, due to its simplicity and yet power to clearly define the major paradigms of asynchronous control circuit behaviour. An STG is a Petri net (PN) whose transitions are labelled with the changes of input and output binary signals. Recently, this model has been generalised so as to avoid overconstraining of the specification domain [12]. This work has also presented a more general structural model,

[1]On leave from: Department of Computing Science, Electrical Engineering University, St. Petersburg 197022, Russia.

Asynchronous Control Structure (ACS). ACS allows multi-output components with possibly non-deterministic behaviour, as opposed to the traditional model of an Asynchronous Logical Circuit (ALC) consisting of single-output components [2]. The corresponding low-level behavioural model, Arc-Labelled Transition System (ALTS), is thus a generalisation of more common State-Transition Diagram (STD). Unlike the latter, whose states are interpreted in terms of binary vectors built on the set of the ALC's signals, ALTS allows *symbolic* representation of the system's states, and transitions between states are not necessarily associated with changes of binary signals. Rather, they can stand for the changes of symbolic states of the abstract components of the ACS.

Developing the idea of behavioural abstraction further, we define an appropriate high-level model called Symbolic Signal Transition Graph (SSTG). We demonstrate some specific properties of this model against "ordinary" STG, and present a synthesis technique, which benefits from the hierarchical representation of circuit behaviour. The latter is defined in terms of an SSTG and a set of local *behavioural cliches* of individual component variables. Such cliches are represented by the variables' State Graphs. During synthesis, these cliches are encoded with binary signals and transformed into local STGs. The global SSTG is then refined using fragments of the individual STGs, thereby generating a global STG expansion. The major advantage of our approach is in providing sufficient conditions for implementability of such an expansion without its explicit verification. According to these conditions, the global implementability can be derived from that of the high-level SSTG specification and those of local behavioural cliches. Building upon stronger and weaker types of implementability we suggest an optimisation strategy that can be applied to gain maximum efficiency from trading off between complexity of the synthesis process and cost of the implementation logic. We illustrate our technique by two synthesis examples: (i) a simple interface controller for a bus slave, and (ii) a control circuit for a bi-directional pipeline channel, which can be used in distributed archtectures with communication networks based on packet switching/routing.

2. BACKGROUND

2.1. Asynchronous Control Structures

The notion of Asynchronous Control Structure (ACS) is a generalisation of the "interconnection structure" of an asynchronous control circuit. It removes the usual structural limitation (e.g., [8]) that each component has exactly one output signal. Thus an ACS can represent an arbitrary interconnection of modules, with the only restriction that no two modules can drive a single signal. An ACS is a directed multigraph (any pair of nodes can be connected by more than one arc) whose nodes designate discrete (i.e., finite-state) components and arcs stand for the interconnections. Every arc is labelled with the name of a *finite-state variable* from a finite set Y ($|Y| = n$). For every variable $y \in Y$, $S^y = \{y^0, y^1, ..., y^k\}$ is called the set of *values*, or states. We also assume that for each variable a specific set of *allowed changes* of values is defined, $D^y \subset S^y \times S^y$, i.e. $D^y = \{(y^i - y^j)|i, j \in 0, 1, ..., k \wedge i \neq j\}$. An ACS is called a *Binary Asynchronous Control Structure* (BACS) if $\forall y : S^y = \{0, 1\}$. Hence, for a BACS, the set of allowed changes can be denoted as $Y \times \{+, -\}$, where "+" stands for a signal change from 0 to 1, and "−" for a signal change from 1 to 0.

The local behaviour of every variable $y \in Y$ is defined by its individual state graph $G^y = \langle S^y, D^y \rangle$: the nodes stand for the values in S^y and arcs for the changes in D^y. Such a graph is called *local behavioural cliche.*

It should be clear that in the normal synthesis process the actual interconnections between components of an ACS can be unknown at the initial stage, which gives the designer freedom to derive them from a high-level behavioural specification.

2.2. Arc-Labelled Transition Systems

The semantics of the behaviour of an ACS is described by an Arc-Labelled Transition System (ALTS), which is a directed graph whose vertices stand for the states (set S) and arcs (set $E, E \subseteq S \times S$) for the transitions of the associated ACS. The transitions are labelled (by function $\gamma : E \rightarrow A$) with the names of actions from alphabet A, which in our case is equal to $\bigcup_{i=1}^{n} D^{y_i}$. Each action represents a change of value of a variable in the ACS, and every path in the graph is a valid sequence of such changes in time. Thus the ALTS describes the complete allowed behaviour of the associated ACS. With each state of an ALTS we associate a symbolic vector consisting of the values of the individual components. I.e., each state is labelled with an element in the cartesian product $\prod_{i=1}^{n} S^{y_i}$. Thus, it is more suitable to define an ALTS as $\langle S, E, \delta \rangle$, where $\delta : S \rightarrow \prod_{i=1}^{n} S^{y_i}$. The labelling of arcs in E by γ in the original ALTS definition can be derived from δ using the information about the allowed transitions, i.e. sets D^{y_i}. For a BACS with a set of variables Y we define a Binary (encoded) Transition System, called State Transition Diagram (STD), in which each state is encoded with a binary vector consisting of the values of Boolean variables, i.e. $\delta : S \rightarrow \{0, 1\}^n$. The i-th component of a state $s \in S$ is denoted as s_i. An ALTS is called *contradictory* if the labelling of states with the value vectors is not injective. Hence, for a non-contradictory ALTS we can identify a state with its label.

For every ALTS(STD) arc (s, s') we allow s and s' to differ in one and only one component, say the i-th. This variable, y_i, is called *excited* in state s and its value s_i is marked with a "*" in s. Since there can be several outgoing arcs from each state, a number of variables can be excited in it. The variables that are not excited in a state are called *stable* in it. We assume that transitions between the states can have *arbitrary but finite delays*, and that these delays are associated with the delays of the components in the modeled ACS (similar to the *gate delay model* in asynchronous circuits). We call an ALTS *initialised* if it has an explicit initial state.

2.3. Symbolic Signal Transition Graphs

An "ordinary" binary Signal Transition Graph (STG) generates an STD as its behavioural semantics [12]. The synthesis process beginning from an STG specification then implies deriving the Boolean function description of the ALC from the corresponding STD, using the existing methods (e.g. [3, 11, 6]). These techniques are essentailly based on the relationship between classes of STG, STD and ALC [12].

As a natural extension of "standard" interpretation of STG, giving rise to BACS or ALC we allow the set of transitions of the underlying PN be labelled with the changes of values in the associated ACS. This helps to model systems in which the "binary image" of abstract *symbolic* values and states has not yet been defined, thus leaving the encoding stage to be a part of synthesis process.

Formally, a Symbolic Signal Transition Graph (SSTG) is a triple $\mathcal{G} = \langle \mathcal{P}, Y, \Delta \rangle$ where $\mathcal{P} = \langle T, P, F, m_0 \rangle$ is a marked Petri Net (PN) (T and P are finite sets of transitions and places, $F \subseteq (T \times P) \cup (P \times T)$ is the flow relation between transitions and places, and m_0 is the initial marking), Y is a set of finite-state variables, each with an associated set of values $S^y, y \in Y$, and $\Delta : T \rightarrow \bigcup_{y \in Y}(\{y\} \times S^y \times S^y)$ labels each transition of $\mathcal{P}$ with a triple "(variable, old value, new value)". An ordinary STG is thus an SSTG whose labelling function is $\Delta : T \rightarrow Y \times \{+, -\}$.

For an SSTG $\mathcal{G} = \langle \mathcal{P}, Y, \Delta \rangle$, the reachability graph generated by its PN $\mathcal{P}$ is an ALTS $\mathcal{S} = \langle [m_0 >, E, \delta \rangle$ such that $[m_0 >$ denotes the set of markings reachable from m_0. For each $m \in [m_0 >$, we have $\delta(m) = s^m$, where s^m is a vector of signal values. Let s_i^m denote the value of y_i in marking m. The ALTS of an SSTG is finite if the underlying PN is *bounded*.

The labelling Δ must be *consistent*, that is for all arcs $e = (m, m')$ in the ALTS:

- if $\Delta(\gamma(e)) = (y_i, v_i, v_i')$, then $s_i^m = v_i*$ and $s_i^{m'} = v_i'$.
- otherwise $s_i^m = s_i^{m'}$.

An SSTG is called *valid* iff its underlying PN is bounded and it has consistent labelling.

Property 2.1 *In a valid* SSTG, *for all firing sequences of its* PN *and every component* $y \in Y$, *the projection of a sequence onto set* D^y *is a sequence of changes allowed by the behavioural cliche of* y.

This property, called *Behavioural Compatibility*, implies that any sequence of actions of the global specification is compatible with the allowed behaviour of all the individual components. In the "binary case", a valid STG generates only firing sequences where the signs of transitions *alternate*.

Thus the validity of an SSTG is *sufficient condition* for being able to generate consistent finite state semantics, which can further be used for implementation.

An SSTG is said to have a *Unique State Coding* (USC) problem if its ALTS is contradictory. The USC condition[2] is necessary for an SSTG to allow deriving its logical implementation.

2.4. Interface Example: Specification

Let us model the behaviour of an abstract ACS shown in Figure 1.(a). It consists of three nodes called bus, register and memory, and a node standing for an interface ("bus slave") controller that generates signals to activate/acknowledge some control actions for the data write operation.

The set of control variables is $Y = \{b, r, m\}$, where b, r and m stand for their corresponding destination nodes: bus, register and memory. Variables b and m are assumed to be binary (states 0 and 1), while r is three-state (0,1 and 2). The interpretation of the transitions of b is as follows: $b+$ denotes the resetting of the output acknowledgement signal, b_a, on the bus handshake and subsequent setting of the input request signal (arrival

[2]Strictly speaking, the USC problem should be lifted to the so-called Complete State Coding problem [3, 6], which takes into account the contradictory states in which only output signals have different enablings.

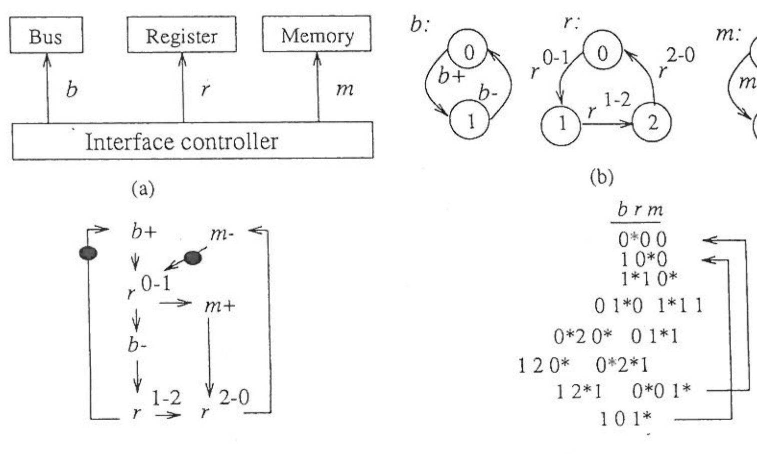

Figure 1. Interface Example: Symbolic Specification

of the new datum), b_r, by the bus master; $b-$ stands for the setting of b_a and subsequent resetting of b_r. For m: $m+$ denotes the setting of the output request, m_r, and subsequent setting of the input acknowledgement, m_a (the datum is written into memory); $m-$ stands for the resetting of m_r and m_a. The interpretation of r, controlling the buffer register, caters for the following requirement. Our aim is to organise a pipeline allowing the bus to carry the new datum as soon as possible, when the register has been loaded with the current value. This captures the idea of *maximal independence* (concurrency) between the bus and the memory handshakes, which are assumed to introduce delays that are relatively larger than that of the register operation. Furthermore, these two delays have least predictable values, and require a speed-independent implementation of the controller. Thus, transition $(r, 0, 1)$ (or r^{0-1}) is similar to the meaning of $m+$ (setting the request r_r and receiving the acknowledgement on r_a), whereas the meaning of r^{1-2} is new. It reflects the situation that the bus, after the reset of b, is ready to issue the next datum, so the register control prepares for the arrival of the new $b+$ by "putting a note for itself" into r. State 2 thus has the meaning: "the new datum on the bus is different from the datum currently stored in the register". The actual reset of the r_r/r_a handshake is performed by transition r^{2-0}. The state graphs for b, m and r are shown in Figure 1.(b). The SSTG in Figure 1.(c) specifies the behaviour of the ACS. It is valid and has the USC property as can be seen from the corresponding ALTS shown in Figure 1.(d).

3. SYNTHESIS APPROACH OUTLINE

We assume that the synthesis process, based on an SSTG as the initial specification, consists of two stages. The first stage constructs the SSTG for an abstract ACS. The result of this stage is an SSTG that is valid and satisfies the USC condition. We call such an SSTG *implementable*. The second stage, on which we shall focus our attention, is concerned with binary encoding of each symbolic variable (unless it is already binary) and then deriving a logical function for each binary signal. For this stage we assume that the binary encoding at the "local" level is more advatageous than the encoding of the global states of the symbolic ALTS. It allows the "local" interpretation of the binary signals associated with the components of the ACS. The second stage of synthesis consists of the following steps:

1. Encode the state graph G^y of each non-binary (i.e. symbolic) variable y with the combinations of values of binary variables $X^y = \{x_1^y, ..., x_k^y\}$; for binary variables y their own values can be used.
2. For each variable y and every transition d^y in $D^y = \{(y^i - y^j) | i, j \in 0, 1, ..., k \wedge i \neq j\}$, construct a partially ordered set (poset) of changes of the encoding variables in X^y, and then build a corresponding fragment of an STG; we denote this fragment $\mathcal{G}^{d^y}$.
3. For each variable y construct an STG $\mathcal{G}^y$ from its state graph G^y and fragments $\mathcal{G}^{d^y}$; in each such STG transitions are labelled with the changes of variables in X^y.
4. Construct a binary STG $\mathcal{G}^*$ from the implementable original SSTG $\mathcal{G}$ by replacing symbolic transitions in the original SSTG with the $\mathcal{G}^{d^y}$ fragments.
5. Check the implementability of STG $\mathcal{G}^*$ and if necessary correct the $\mathcal{G}^{d^y}$ fragments.
6. Derive the Boolean functions for each output binary variable in $\bigcup_{i=1}^n X_i^y$.
7. Construct the output signals from outputs in $\bigcup_{i=1}^n X_i^y$ and connect them with the controlled components in the ACS.

We should now derive the implementability conditions for STG $\mathcal{G}^*$.

4. IMPLEMENTABILITY OF BINARY EXPANSION

4.1. Binary Expansion Construction

For every variable $y \in Y$, whose local behaviour is defined by state graph $G^y = \langle S^y, D^y \rangle$, we define a set of binary encoding signals $X^y = \{x_1, ..., x_k\}$. It is clear that any encoding method will have $k \geq \lceil \log |S^y| \rceil$. We define an *encoding function* λ, consisting of two parts, λ^S and λ^D, as follows:

- $\lambda^S : S^y \to \{0,1\}^k$ is injective;
- $\lambda^D : D^y \to 2^{(X \times \{+,-\})}$, such that $\forall d = (s_1, s_2) \in D^y$:
$\lambda^D(d) = \{(x_j, (\lambda_1 - \lambda_2)_j) | 1 \leq j \leq k \wedge (\lambda_1 - \lambda_2)_j \neq *\}$, where

$$(\lambda_1 - \lambda_2)_j = \begin{cases} + & \text{if } \lambda_{1_j} = 0 \wedge \lambda_{2_j} = 1 \\ - & \text{if } \lambda_{1_j} = 1 \wedge \lambda_{2_j} = 0 \\ * & \text{if } \lambda_{1_j} = \lambda_{2_j} \end{cases}$$

in which $\lambda_1 = \lambda^S(s_1)$ and $\lambda_2 = \lambda^S(s_2)$.

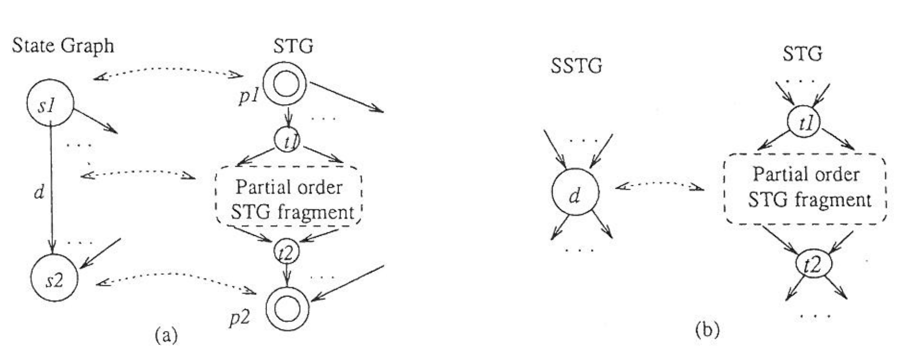

Figure 2. Binary Expansion at the local (a) and global (b) levels

Therefore λ^D assigns to each arc d in G^y a subset of variables (called *transition subset*), together with the "signs" of their changes, whose values change between s_1 and s_2.

The complete binary expansion of each behavioural cliche also requires defining, for each d, a partial order over the corresponding transition set $\lambda^D(d) \subseteq X^y$. It is obvious that the optimal, in terms of speed, solution would allow for maximally parallel switching of all elements in each transition subset. This, however, is not always acceptable as some intermediate states occurring between the main states in S^y may coincide in their encodings. This may result in violation of the USC condition for the final STG.

For every poset built on sets $\lambda^D(d)$ we can constract a (fragment of) STG,$\mathcal{G}^{d^y}$, and make the following transformation of the state graph G^y for each y. Each state in G^y forms a PN place and for each arc d we create two auxiliary *dummy* (not labelled through Δ) transitions, between which the STG $\mathcal{G}^{d^y}$ is inserted as shown in Figure 2.(a).

Avoiding unnecessary formalism, we state that by refining each arc d of G^y in the above way we can always build a corresponding STG $\mathcal{G}^y$. The initial marking of this STG can be easily found from the knowledge of the value of y in the initial state of the original SSTG $\mathcal{G}$: the place associated with this state, s_0^y, is assigned with marking $m(s_0^y) = 1$.

The following property holds due to the above construction and the fact that each variable has some initial state consistent with the initial marking of $\mathcal{G}$.

Property 4.1 *The* STG *built for each variable's cliche is consistent and its underlying* PN *is a safe free-choice* PN[3].

This property implies that each local STG is valid.

The transitions of variables in Y can also be refined at the global level, in the original SSTG $\mathcal{G} = \langle \mathcal{P}, Y, \Delta \rangle$ into an STG $\mathcal{G}^* = \langle \mathcal{P}^*, X, \Delta^* \rangle$, where X is the union-set of all encoding variable sets X^y. Thus each labelled (with some d) transition $t \in T$ in the underlying PN $\mathcal{P} = \langle P, T, F, m_0 \rangle$ we substitute with STG $\mathcal{G}^{d^y}$ as shown in Figure 2.(b). The global binary STG satisfies the following.

Proposition 4.1 *The global* STG *expansion of a valid* SSTG *is also valid iff all the local cliches'* STG *expansions are valid.*

[3]The definitions of *free-choice, safe* etc. Petri Nets can be found for example in [9].

4.2. Implementability Conditions

4.2.1. Strong Implementability

Since validity of an STG is not sufficient for its implementability we need to impose some more restrictions on the STG's of variables in Y.

For each variable $y \in Y$ its local STG G^y generates an STD $\mathcal{S}^y$ in the usual way. Each such STD is called the *state-transition expansion* of the variable's state graph G^y. Due to the construction technique, $\mathcal{S}^y$ contains all the states that G^y has (encoded by the corresponding λ^S), which are called *nodal* states, plus some extra states, called *transient*, generated as a result of the *interleaving semantics* of the concurrent transitions of signals in the transition subsets.

The following sufficiency condition is called *Strong Implementability*.

Claim 4.1 *The global* STG *expansion of an implementable* SSTG *is also implementable if all the variables'* STG *expansions are implementable.*

Proof Hint. Due to Proposition 4.1 we need only to check if the USC conditions holding true for the SSTG and all the local STGs imply the USC for the global STG. This is easily proven by considering the STD generated by the global STG as a state-transition expansion of the non-contradictory ALTS, and by the rules we applied for encoding the symbolic values of the original variables by the combinations of binary signals.

This claim is important since it helps to reduce the problem of checking the global STG's implementability to a number of subproblems of smaller size, thereby gaining from our description hierarchy. Unfortunately this approach, despite its algorithmic efficiency, may be too restrictive. While ensuring the implementability of each local STG, we may overly constrain concurrency of the transitions of variables in X^y. It is intuitively clear that some local USC violations can be "compensated" by using the distinguishing capability of the combinations of values of the remaining variables, $Y \setminus y$. On the other hand, since such values are also binary encoded, finding a general solution, which would provide *sufficient and necessary* conditions, can be algorithmically hard. It would thus be desirable to restrict ourselves with a "softer" approach. Among the distingishing variables we shall only consider those ones whose values in the states "plagued" by USC violations are *stable*, so we can fully benefit from staying at the symbolic level.

4.2.2. Weak Implementability

First, let us formally define the notion of indirectly distinguishable states of a symbolic variable y. Let $S1 \subseteq S$ ($S2 \subseteq S$) be the set of states of the ALTS $\mathcal{S} = \langle S, E, \delta \rangle$ generated by the original SSTG $\mathcal{G}$ such that in all states in $S1(S2)$ component y has the same value u_1 (u_2). Note that u_1 and u_2 are taken with their excitation label (let us call them *full values*), and if y is excited in some state s in $S1$ ($S2$) it must be excited in all states in $S1$ ($S2$). So, if for example y has the value of v_1 and is excited in $S1$, we say $u_1 = v_1^*$. It is clear that $S1 \cap S2 = \emptyset$. Let $s1 \in S1, s2 \in S2$. We denote, by $\delta(s \setminus y)$, the vector of values of all but y components in Y associated with a state $s \in S$. Let $diff(s1 \setminus y, s2 \setminus y)$ denote the set of variables in $Y \setminus \{y\}$ whose values are *stable and different* in their labellings $\delta(s1 \setminus y)$ and $\delta(s2 \setminus y)$.

For full values u_1 and u_2 of y we construct a characterstic Boolean function, $\mathcal{F}(u_1, u_2)$, called *Distinction Function*, such that $\mathcal{F}(u_1, u_2) = 1$ if there exists at least one pair of

states $s1 \in S1, s2 \in S2$ ($s1, s2 : s1^y = u_1$ and $s2^y = u_2$) such that $diff(s1 \setminus y, s2 \setminus y) \neq \emptyset$, otherwise $\mathcal{F}(u_1, u_2) = 0$.

Now, assume Strong Implementability does not hold: for some a variable $y \in Y$ its STG has USC problem, i.e. its associated STD $\mathcal{S}^y = \langle \Sigma^y, E^y, \delta^y \rangle$ is contradictory. Thus, there is at least one pair of states σ_1 and σ_2 in Σ^y such that $\delta^y(\sigma_1) = \delta^y(\sigma_2)$. Let us call any such pair of states *locally indistinguishable*. It is obvious that for every nodal state σ in $\mathcal{S}^y$, δ^y is related to the encoding function λ^{S^y}, applied to the state graph G^y, in the following way: $\delta^y(\sigma) = \lambda^{S^y}(\sigma)$. Due to this, the above two states σ_1 and σ_2 cannot be nodal since λ^{S^y} is injective by construction. Therefore, we have two remaining cases: (1) σ_1 is transient and σ_2 is nodal, and (2) both σ_1 and σ_2 are transient.

Consider only the first case (the case with transient σ_2 and nodal σ_1 is symmetrical). The second one can be treated in the same way.

By our construction of the local STG, in the generated STD $\mathcal{S}^y$ there must be a *unique* nodal state, $\bar{\sigma}_1$, which is the *nearest nodal predecessor* of σ_1. σ_1 occurs in the transition of y from the state corresponding to $\bar{\sigma}_1$ to some other nodal state. By the above relationship between δ^y and λ^{S^y} for nodal states, we can uniquely derive the values of y in the original state graph G^y, $v_1 = (\lambda^{S^y})^{-1}(\delta^y(\bar{\sigma}_1))$ and $v_2 = (\lambda^{S^y})^{-1}(\delta^y(\sigma_2))$. Since $\bar{\sigma}_1$ is the nearest nodal predecessor of σ_1, the latter should correspond to the same symbolic value v_1 but with the $*$ symbol. Thus, our states σ_1 and σ_2 are called *indirectly distinguishable* if $\mathcal{F}(v_1^*, v_2) = 0$ and $\mathcal{F}(v_1^*, v_2^*) = 0$.

For the case when both σ_1 and σ_2 are transient, the latter condition transforms into $\mathcal{F}(v_1^*, v_2^*) = 0$, where v_2 is obtained in analogous way, i.e. using the nearest nodal prdecessor for σ_2.

Claim 4.2 *The global* STG *expansion of an implementable* SSTG *is also implementable if either all the variables'* STG *expansions satisfy* USC *condition or for each variable y whose* STG *generates the* STD *with locally indistinguishable states, all such states are pairwise indirectly distinguishable.*

Proof Hint. Any transient state $\sigma \in \Sigma$ has a unique nearest nodal predecessor $\bar{\sigma}$ and $diff(s1 \setminus y, s2 \setminus y)$ is preserved in any firing sequence where y changes between $\bar{\sigma}$ and σ.

This condition is called *Weak Implementability*. Unfortunately, even this condition is only sufficient and could be more relaxed. However, this would involve a much more thorough analysis of distinguishability, e.g. by modifying Distinction Function to cater for the difference not at the "symbolic" level but at the level of binary codes of the remaining (other than y) variables. The search of that type, when converted into an algorithmic procedure, would no longer benefit from the hierarchical description provided by SSTG.

The above approach suggests two useful strategies for a constrained optimisation:

- Begin with Strong Implementability, i.e. construct a combination of an implementable SSTG and locally implementable STG's of variables in Y. Then lift some order constraints at the local cliche level, by allowing up to maximum concurrency between transitions of the encoding variables in λ^{D^y} for all y, until Weak Implementability no longer holds.

- Begin with the maximally concurrent local STGs, and while adding new ordering constraints in λ^{D^y} for all y, check when Weak Implementability becomes true.

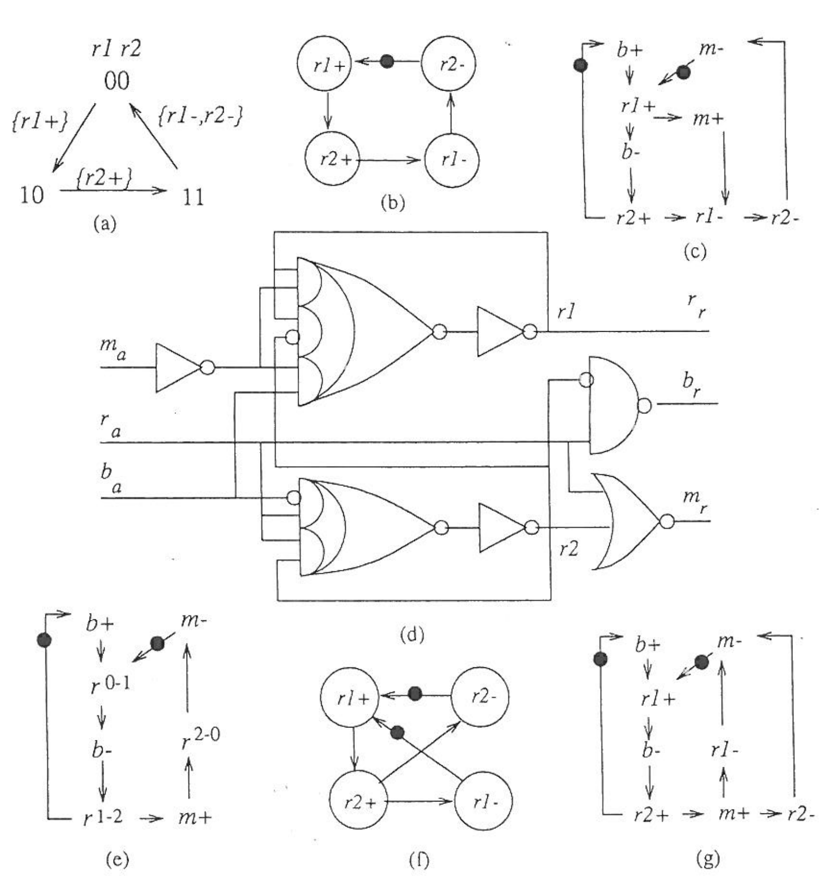

Figure 3. Interface Example: Synthesis

4.3. Interface Example: Circuit Synthesis

In this section we use the above technique to synthesise two implementations of the SSTG shown in Figure 1.(c) Since among the variables in $Y = \{b, r, m\}$ only r is non-binary, we have to encode it with auxiliary binary signals, r_1, r_2. The result of encoding the state graph G^r (see Figure 1.(b)) is shown in Figure 3.(a).

Here, $\lambda^D(r^{0-1}) = \{r_1+\}, \lambda^D(r^{1-2}) = \{r_2+\}$. In order to satisfy Strong Implementability (it is trivially holds for b and m) we create the following partial order for $\lambda^D(r^{2-0}) = \{r_1-, r_2-\} : r_1- \leq r_2-$. The STG $\mathcal{G}^r$ for r is shown in Figure 3.(b). The global binary STG $\mathcal{G}^*$ is shown in Figure 3.(c). It is easily seen that this STG is implementable. Using one of the existing methods [10, 3, 6] for ordinary STG we obtain the following set of Boolean functions: $b = \bar{r_1} + r_2, m = r_1 + r_2, r_1 = \bar{m} + (\bar{m} + \bar{r_2})r_1, r_2 = \bar{b}r_1 + r_1r_2$ The circuit implementing the interface controller component in Figure 1.(a) is shown in Figure 3.(d). The appropriate "request/acknowledgement" pairs of handshake signals are

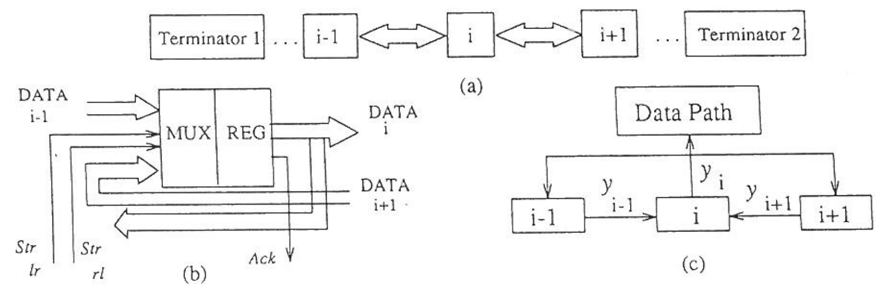

Figure 4. Two-way Pipeline: Structural View

formed by breaking the interconnections of signals b, r_1 and m.

If we try to avoid putting r_1- and r_2- in sequence, then the $\mathcal{G}^r$ with concurrent transitions r_1- and r_2- will not be implementable since the STD generated by it will contain two states labelled with 10. One, nodal state, is reachable from the initial state, 00, by the firing of r_1+. The other, transient state, is reachable from the nodal state 11 if r_2- fires ahead of r_1-. Unfortunately, by checking Weak Implementability we find that these two states are not indirectly distinguishable: the pair of symbolic states labelled as 01*1 and 02*1, respectively, in Figure 1.(d) cannot distinguish the above two 10 states of r by their stable component (here, $m = 1$). This situation results in the violation of implementability of the corresponding global binary STG.

As an example where Weak Implementability has its effect, consider the SSTG shown in Figure 3.(e). This specification is less concurrent at the symbolic level than the original one. However, the violation of the USC condition at the local level (for r), which can be seen from the local STG in Figure 3.(f), does no longer preclude the implementation of the global binary STG expansion shown in Figure 3.(g). The latter produces the following logical functions: $b = m + \bar{r_1} + r_2, m = mr_1 + r_2, r_1 = b\bar{m} + \bar{m}r_1, r_2 = \bar{b}\bar{m}r_1 + \bar{m}r_2$.

5. SYNTHESIS OF TWO-WAY PIPELINE CONTROLLER

In this section we show how SSTG and the above technique can be used to synthesise a control circuit for a bidirectional pipeline communication channel. Such a channel can be organised in a massively parallel system where the modules communicate through a packet switching/routing network. We assume that the channel has a data path which allows data to be transferred in two directions in a time-shared way. The structural description is shown in Figure 4, where three adjacent cells, $(i-1)$-th, i-th and $(i+1)$-th, are singled out for convenience. When the pipeline is in the left-to-right ("→"or "lr") mode a datum is shifted into the register (REG) of the i-th cell from its left neighbour cell. This is controlled by a strobe signal Str_{lr}, using a multiplexor (MUX). When the pipeline is in the right-to-left ("←" or "rl") mode a datum is shifted from the right neighbour cell, with the aid of another strobe, Str_{rl}.

Our goal is to implement the controller of the i-th cell, which will generate the strobe signals depending on the direction mode set in the controller during special "mode change sessions". Since we assume the pipeline to operate in a totally distributed way, the cells

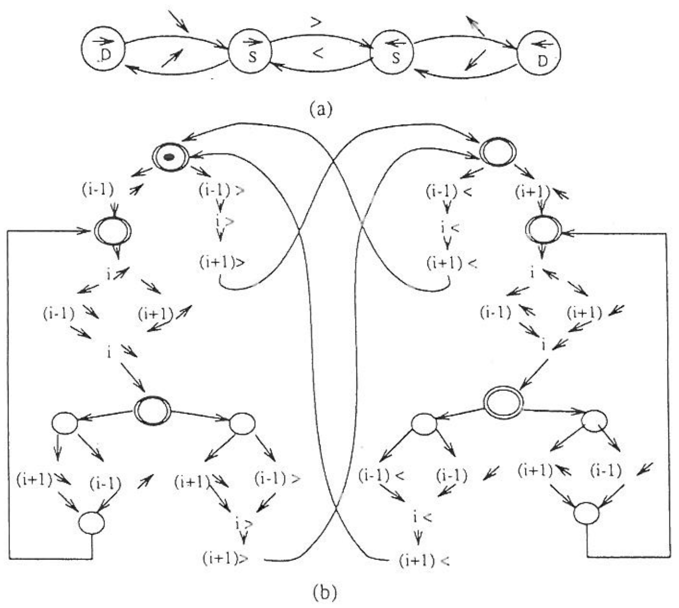

Figure 5. First idea: State Graph (a) and SSTG (b)

are informed about a particular mode through the same pipeline. To facilitate this we arrange that once data is being sent by one terminator, only this terminator owns the *privilege* to transmit data, to stop the transmission and to hand over the privilege to the other terminator. The latter, after using the channel in similar way, may send the privilege back to the first terminator and the process repeats.

Figure 4.(c) shows the basic ACS for synthesis, in which variables y_{i-1} and y_{i+1} are inputs for the i-th cell while y_i is the output, which can be used by the neighbouring cells and the data path of the i-th cell. Thus $Y = \{y_{i-1}, y_i, y_{i+1}\}$.

5.1. First Idea

Since the local cliches of all variables in Y should be of the same type (pipeline regularity), only a generic state graph for just one variable, say y_i, is shown in Figure 5.(a). The variable has four states: $S^y = \{\overrightarrow{S}, \overrightarrow{D}, \overleftarrow{S}, \overleftarrow{D}\}$, where $\overrightarrow{S}$ ($\overleftarrow{S}$) stands for the spacer in the $\rightarrow$ ($\leftarrow$) mode, and $\overrightarrow{D}$ ($\overleftarrow{D}$) for the data valid state in the $\rightarrow$ ($\leftarrow$) mode.

Corresponding notation is used to model the allowed changes of values of the variable: $D^y = \{\nearrow, \searrow, \nwarrow, \swarrow, >, <\}$.

The interaction involving the three adjacent cells is defined by the SSTG shown in Figure 5.(b). In this graph the transitions are labelled with the identifier of the cell variable and the corresponding value change, taken from the variable's cliche. The initial marking is the state in which all the cells are in the $\overrightarrow{S}$, the privilege is with the leftmost cell

and the latter determines whether to initiate the data transmission or send the privilege to the other end by changing its state to $\overleftarrow{S}$. If the first option is chosen, the leftmost cell changes to $\overrightarrow{D}$, and the whole pipeline behaves like an ordinary one-way pipeline until the leftmost cell, while being in $\overrightarrow{S}$, decides to hand over the privilege by changing its state to $\overleftarrow{S}$. The i-th cell, when it accepts $\overleftarrow{S}$ from its left neighbour, also changes from $\overrightarrow{S}$ to $\overleftarrow{S}$. When the privilege reaches the rightmost cell, the latter begins either transmitting data or hands the privilege back to its left neighbour. It is easy to see that the behaviour will now be similar to the above except for the roles of "right" $(i+1)$ and "left" $(i-1)$ having been interchanged.

The SSTG is valid (it is consistent and its PN is bounded), however it appears to have a USC problem, which could be seen from the ALTS generated by the SSTG (due to space limitation it is not shown here and we leave at as an exercise). This ALTS is contradictory and has two pairs of different states with the same labelling. For one such pair, the output variable y_i is unable to determine from the state of the other two variables, whether to remain stable (state $\overleftarrow{S}\ \overleftarrow{S}\ \overrightarrow{S}^*$) or become excited (state $\overleftarrow{S}\overleftarrow{S}^*\ \overrightarrow{S}$). As a consequence, even our SSTG is not implementable, regardless of the implementability of the local behaviour of variables.

5.2. Second Idea: Correct Solution

The major problem of our first attempt was in the incapability of variable y_i to indicate (for itself) the fact of the change of the transmission direction. This can be fixed by adding a pair of new states (one for each direction), $\overrightarrow{C}$ and $\overleftarrow{C}$, to the local behavioural cliches of the variables in Y, which would mark the change of the transmission mode. Such a state can be interpreted as an explicit "privilege" transmitted through the channel. The cliche's state graph is shown in Figure 6.(a). With the pair of new states we have also obtained the new changes, $>1, >2, <1$ and <2, whose meaning should be obvious.

The SSTG satisfying the above cliche is shown in Figure 6.(b). This graph has no USC problem and, since being valid, is implementable (the reader may again try to build its ALTS as an exercise).

We proceed further with the binary encoded state graph shown in Figure 6.(c). The set of encoding variables is $X^{y_i} = \{a_i, b_i, c_i\}$. This graph gives rise to the local STG, shown in Figure 6.(d), which is in this case uniquely built from the state graph since all the transition sets are singletons, and there is no choice on how to order the binary signal changes. The global binary STG can be easily built from the SSTG and the encoded transitions of variables. For example, $(i-1)\nearrow$ is substituted with the transition labelled with $a_{i-1}+$, $(i-1)>$ with $b_{i-1}+$ and so on.

Since the local cliches are also implementable, Claim 4.1 implies that the global STG is implementable too. Using a standard technique [10, 3, 6] we obtain the logical implemenation for output signals, a_i, b_i and c_i:

$$\begin{aligned}
a_i &= a_{i-1}\bar{a}_{i+1}\bar{c}_i + \bar{a}_{i-1}a_{i+1}c_i + a_i(\bar{a}_{i-1}c_i + a_{i-1}\bar{c}_i + a_{i+1}c_i + \bar{a}_{i+1}\bar{c}_i) \\
b_i &= a_{i-1}c_i + \bar{a}_{i+1}b_{i-1}\bar{c}_i + b_{i+1} \\
c_i &= b_{i-1}c_i + b_{i+1}c_{i-1} + c_{i+1}
\end{aligned}$$

Using the semantics of the local cliche and state encoding it is now easy to construct the output strobe signals: $Str_{lr} = a_i\bar{b}_i$ and $Str_{rl} = a_ib_i$.

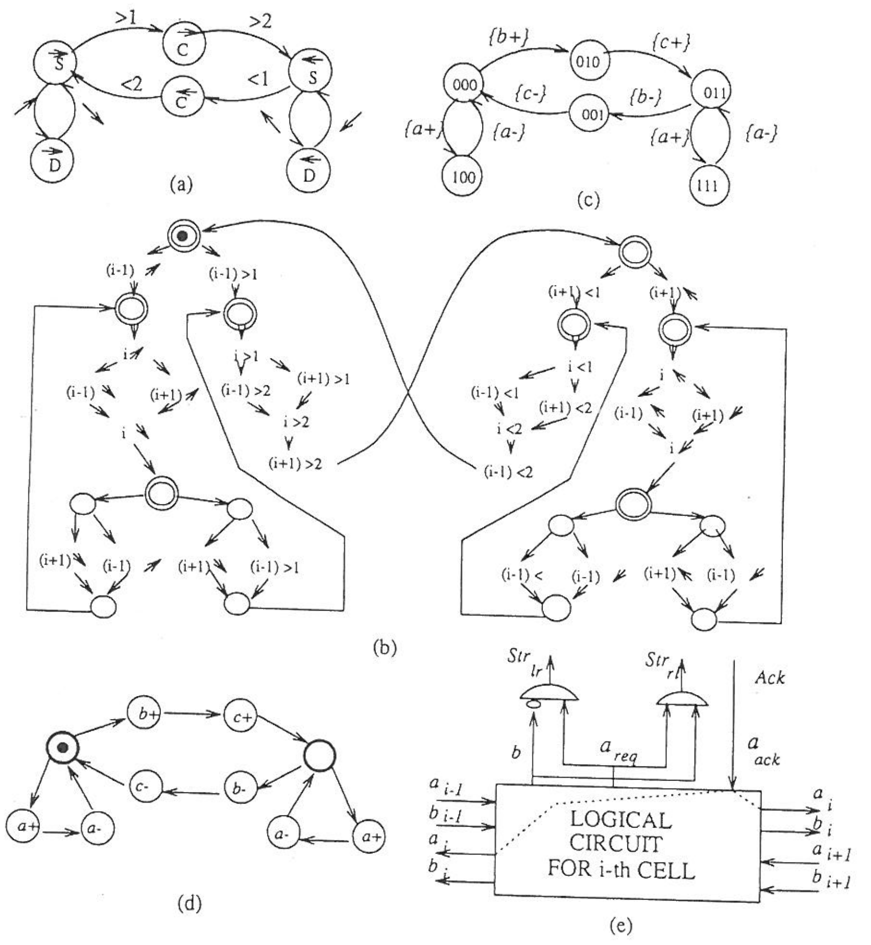

Figure 6. Correct Solution

The actual interconnection of the abstract ACS in Figure 4.(c) is shown in Figure 6.(e). In order to use signal a_i one has to break the output wire from the gate implementing a_i, and connect it to the signal labelled as a_{req}. This signal is used to strobe the AND gates for Str_{lr} and Str_{rl}. The *Ack* input produced by the data path logic should be connected to the other end of the broken wire (a_{ack}), and then used for the external interconnections of the i-th cell. It is easily seen from the local STG that signal b_i can be directly used at the ANDs because it is always stable when the circuit changes the value of a_i.

6. CONCLUSIONS

With the proposed synthesis approach, based on a high-level specification language, Symbolic Signal Transition Graph, we have achieved: (i) a better way of capturing the system's abstraction and hierarchy, and (ii) more efficient checking of its implementability conditions. The circuit designer would maximally benefit from this approach if he/she would be able to describe the initial idea of the synthesised Asynchronous Control Structure in an essentially abstract form. The number of components should be kept relatively small, while their behaviour be specified as a multi-state (not necessarily two-state like in "ordinary" Signal Transition Graph) graph that is further refinable by binary encoding. The sufficient conditions of the implementability of a binary expansion of the global specification suggest flexible optimisation strategies.

Acknowledgements:

The authors would like to thank Luciano Lavagno for his useful comments and the British Royal Society for supporting A. Petrov's attendance at the Conference.

REFERENCES

1. K. van Berkel. *Handshake circuits: an intermediary between communicating processes and VLSI.* PhD thesis, Technical University of Eindhoven, 1992.
2. J.A. Brzozowski and C.-J.H. Seger. Advances in asynchronous circuit theory. *Bulletin of the EATCS*, (42), October 1990. (Part 2 in No.43, Feb. 1991).
3. T.A. Chu. *Synthesis of Self-Timed VLSI Circuits from Graph-theoretic Specifications.* PhD thesis, MIT, 1987.
4. M. Josephs and J.T. Udding. An algebra for delay-insensitive circuits. Technical Report WUCS-89-54, Dept of CS, Washington University, St Louis, 1989.
5. M.A. Kishinevsky, A.Y. Kondratyev, and A.R. Taubin. Formal method for self-timed design. In *Proceedings of EDAC'91*, pages 197–201, 1991.
6. L. Lavagno, K. Keutzer, and A. Sangiovanni-Vincentelli. Algorithms for synthesis of hazard-free asynchronous circuits. In *Proceedings of the DAC'91*, June 1991.
7. A.J. Martin. Synthesis of asynchronous vlsi circuits. In J. Staunstrup (ed.), editor, *Formal Methods for VLSI Design*, chapter 6. North Holland, Amsterdam, 1990. IFIP WG 10.5 Lecture Notes.
8. R.E. Miller. *Switching Theory*, volume 2: Sequential Circuits and Machines, chapter 10. Wiley-Interscience, New York, 1965.
9. T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of IEEE*, 77(4):541–580, April 1989.
10. L.Ya. Rosenblum and A.V. Yakovlev. Signal graphs: from self-timed to timed ones. In *Proceedings of International Workshop on Timed Petri Nets, Torino, Italy, July 1985*, pages 199–207. IEEE Computer Society, 1985.
11. V.I. Varshavsky, editor. *Self-Timed Control of Concurrent Processes.* Kluwer AP, Dordrecht, 1990.
12. A. Yakovlev, L. Lavagno, and A. Sangiovanni-Vincentelli. A unified STG model for asynchronous control circuit synthesis. In *Proceedings of ICCAD'92*, Santa Clara, CA, November 1992.