Symbolic Signal Transition Graphs and Asynchronous Circuit Design

A.V. Yakovlev and A.I. Petrov

Abstract

Signal Transition Graph (STG), the interpretation of Petri Net (PN) by the changes of values of binary signals, has recently become popular as a model for asynchronous control circuit synthesis. It is however unable to define the behaviour of abstract Asynchronous Control Structures (ACSs), whose components have many outputs and "signals" are multi-valued or symbolic.

Our aim is to present a model, called Symbolic Signal Transition Graph (SSTG), that is the 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, effectively illustrate the approach.
Bibliographical details

YAKOVLEV, Alexandre Vladimir
Symbolic Signal Transition Graphs and Asynchronous Circuit Design.
Newcastle upon Tyne: University of Newcastle upon Tyne: Computing Laboratory, 1992.
(University of Newcastle upon Tyne, Computing Laboratory, Technical Report Series, no. 395)

Added entries
UNIVERSITY OF NEWCASTLE UPON TYNE.
Computing Laboratory. Technical Report Series.395
PETROV, Alexei Igor

Abstract

Signal Transition Graph (STG), the interpretation of Petri Net (PN) by the changes of values of binary signals, has recently become popular as a model for asynchronous control circuit synthesis. It is however unable to define the behaviour of abstract: Asynchronous Control Structures (ACSs), whose components have many outputs and "signals" are multi-valued or symbolic.

Our aim is to present a model, called Symbolic Signal Transition Graph (SSTG), that is the 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, effectively illustrate the approach.

About the author

Dr. A.V. Yakovlev joined the Department of Computing Science as a Lecturer in September 1991, after a year as a Senior Lecturer at the Polytechnic of Wales, where he had been on leave from the Electrical Engineering Institute at Saint Petersburg, Russia.

Dr. I.A. Petrov, an Assistant Professor at the Department of Computing Science of the Electrical Engineering Institute at Saint Petersburg, Russia. During 1990-92 he has been a visiting researcher at the Helsinki University of Technology and Swedish Institute of Technology at Helsinki, Finland. He visited the Department of Computing Science at Newcastle in August 1992.

Suggested keywords

ASYNCHRONOUS CONTROL CIRCUITS LOGICAL SYNTHESIS PETRI NETS
HARDWARE DESIGN AUTOMATION SIGNAL TRANSITION GRAPHS

Suggested classmarks (primary classmark underlined)
Dewey (18th): 621.38173 001.64404
U.D.C. 621.3.049.771.14 519.687
Symbolic Signal Transition Graphs and Asynchronous Circuit Design

Alexandre V. Yakovlev
Department of Computing Science
University of Newcastle upon Tyne

Alexei I. Petrov *
Department of Computing Science
Electrical Engineering Institute
St. Petersburg, 197022, Russia

Abstract

Signal Transition Graph (STG), the interpretation of Petri Net (PN) by the changes of values of binary signals, has recently become popular as a model for asynchronous control circuit synthesis. It is however unable to define the behaviour of abstract Asynchronous Control Structures (ACSs), whose components have many outputs and "signals" are multi-valued or symbolic.

Our aim is to present a model, called Symbolic Signal Transition Graph (SSTG), that is the 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, effectively illustrate the approach.

1 Introduction

With increasing interest in system-level VLSI design asynchronous circuits are becoming a crucial part of intercomponent interfaces and various control-dominated structures. The two major steps that the circuit designer has to overcome are: (i) specifying in a clear way, preferably in a sufficiently abstract manner, the behaviour of the system; and (ii) implementing this behaviour correctly by an interconnection of primitive components, e.g. logical gates. A number of formal specification languages and synthesis methodologies have been proposed in [12, 4, 9, 5, 1, 6].

Among them, Signal Transition Graph (STG), a model based on interpreted Petri nets (PNs), has acquired high popularity, due to its simplicity and yet power to clearly define the major paradigms of asynchronous control circuit behaviour. An STG is a PN whose transitions are labelled with the changes of input and output binary signals. Recently, this model, whose original definition [4] appeared to be too restrictive [16], has been generalised so as to avoid any danger of unnecessary constraining the designer’s specification domain [15]. This work has provided not only a broader view on the behavioural paradigms of asynchronous control circuits but also presented a more general structural description of the design circuit. This model, called Asynchronous Control Structure (ACS), allows multi-output components, as opposed to the traditional model of an Asynchronous Logical Circuit (ALC) consisting of single-output components [2]. The corresponding low-level behavioural model, called Arc-Labelled Transition

*During August 1992 visited University of Newcastle, Department of Computing Science
System (ALTS) is thus a generalisation of a more common State-Transition Diagram (STD). Unlike the latter, whose states are interpreted in terms of binary vectors built on the set of the ALCs signals, an 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.

Further to this idea of behavioural abstraction, we define an appropriate high-level model, called Symbolic Signal Transition Graph (SSTG) \(^1\). We demonstrate some specific properties of this model against “ordinary” STG, and present a synthesis technique, which uses the benefits of hierarchical representation of the 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 by 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. These conditions imply the way how global implementability can be derived from that of the high-level SSTG specification and those of local behavioural cliches.

We build upon two types of implementability, called Strong and Weak Implementability, and suggest an adequate 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 with two synthesis examples. The first example is a simple interface controller for a bus slave. The second example is a control circuit for a bi-directional pipeline channel, which can find its application in highly parallel, distributed architectures, such as for example communication networks based on packet switching/routing.

The overall organisation of the paper is as follows. Section 2 gives a brief formal background, introducing the main concepts (ACS, ALTS and SSTG) and notation. Section 3 outlines our synthesis approach. Section 4 derives the implementability conditions. Section 5 describes the synthesis of a two-way pipeline controller.

## 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., [10] or [13]) 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. The formal definition of a discrete ACS is in [15]. Informally an ACS is a directed graph whose nodes designate discrete (i.e. finite-state) components, and whose 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 variable values, or states. We also assume that for each variable a specific set of allowed changes of values is defined, \(D^y \subseteq S^y \times S^y\), i.e. \(D^y = \{(y^i - y^j) | i, j \in 0, 1, ..., k \land i \neq j\}\). 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.

\(^1\)The idea of using this term belongs to L. Lavagno [7].
A convenient way of defining the local behaviour of a variable \( y \in Y \) is its individual state graph \( G^y = (S^y, D^y) \): 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 is crucial to note that in the normal synthesis process the actual interconnections between components of an ACS can be unknown at the initial stage, thus giving 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), in [15]. An ALTS 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 latter are labelled (by function \( \gamma : E \rightarrow A \)) with the names of actions from alphabet \( A \), which in our case is equal to \( \bigcup_{y=1}^{n} D^y \). Each action represents a change of value of a variable in the ACS, and every path along the graph represents 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 \( \prod_{y=1}^{n} S^y \). Thus, it is more suitable to define an ALTS as \( (S, E, \delta) \), where \( \delta : S \rightarrow \prod_{y=1}^{n} S^y \). The labelling of arcs in \( E \) by \( \gamma \) in the original ALTS definition can be derived from \( \delta \) using the information about the allowed transitions, i.e. sets \( D^y \). 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 the vector associated with each 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, connecting a pair of states \( s \) and \( s' \), 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 as its behavioural semantics an STD [15]. 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. [4, 14, 8, 17]). These techniques are essentially based on the relationship between classes of STG, STD and ALC summarised in [15].

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 of the components of the associated ACS. This helps to model systems whose “binary image” of a more abstract symbolic values and states has not yet been defined yet, thus leaving the encoding stage as a part of an optimal synthesis process.
Formally, a Symbolic Signal Transition Graph (SSTG) is a triple $G = (P, Y, \Delta)$ where $P = (T, P, F, m_0)$ 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 $P$ with a triple "(variable, old value, new value)". An ordinary STG is thus an SSTG whose labelling function is $\Delta : T \rightarrow Y \times \{+, -, \}$.

Given an SSTG $G = (P, Y, \Delta)$ and the reachability graph generated by its underlying PN $P$, we define a labelled reachability graph, or ALTS, as $S = (\{m_0 >\}, E, \delta)$, where $\{m_0 >\}$ denotes the set of markings reachable from $m_0$, as follows. For each $m \in \{m_0 >\}$, we have $\delta(m) = s^m$, where $s^m$ is a vector of signal values. Let $s^m_i$ denote the value of signal $y_i$ in marking $m$. The ALTS of an SSTG can be constructed assuming that the underlying PN is bounded.

The labelling $\Delta$ must be consistent, so we must have for all arcs $e = (m, m')$ in the ALTS:

- if $\Delta(\gamma(e)) = (y_i, v_i, v'_i)$, then $s^m_i = v_i$ and $s^{m'}_i = v'_i$.
- otherwise $s^m_i = s^{m'}_i$.

An SSTG is called valid iff its underlying PN is bounded (hence its ALTS is finite) 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, manifests 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\textsuperscript{2}, is necessary for an SSTG to allow deriving its logical implementation.

### 2.4 Interface Example: Specification

We illustrate the above formal background with a simple example of a bus-register-memory slave interface ("write" operation) specification borrowed from [16]. We 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 the interface controller that generates signals to activate/acknowledge some control actions.

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 following: $b+$ denotes the resetting of the output acknowledgement signal, $b_0$, on the bus handshake and subsequent setting of the input request signal (arrival of the new datum), $b_1$, by the bus master;

\textsuperscript{2}Strictly speaking, the USC problem should be lifted to the so-called Complete State Coding problem [4, 8], which takes into account the contradictory states in which only output signals have different enablings.
Figure 1: Interface Example: Symbolic Specification

\( b^- \) stands for the setting of \( b_a \) and subsequent resetting of \( b_a \). 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 capable of 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 \textit{maximally independent} (concurrent) operation of the two handshake links, bus and memory, which are assumed to introduce delays that are relatively larger than that of the register operation. Furthermore these two delays have most unpredictable 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 \( r \), 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 change \( b+ \) by "putting a note for itself" into \( r \). State 2 has the meaning "the new datum on the bus is different from the datum currently stored in the register". The actual reset of the handshake \( r_r / r_a \) is performed by the third 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 The 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 \textit{implementable}. The second stage is concerned with introducing a binary encoding for the values of each symbolic
variable (unless it is already binary) and then deriving a logical function implementation for each binary signal. For this stage we assume that the binary encoding at the "local" level rather than encoding of the global states of the symbolic ALTS is the designer's requirement. It is caused by the "local" interpretation of the binary signals associated with the components of the ACS.

The second stage of synthesis, on which we put emphasis here, consists of the following steps:

1. Encode the state graph \( G^y \) of each non-binary (i.e. symbolic) variable with the values of binary variables \( X^y = \{ x_1^y, ..., x_k^y \} \), using one of the known (in theory of asynchronous FSM, e.g. [13]) methods; for binary variables \( y \) their own names can be used.

2. For each variable, \( y \), and every transition, \( d^y \) of this variable in \( D^y = \{ (y^i - y^j) | i, j \in 0, 1, ..., k \land i \neq j \} \), 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 \( G^{d^y} \).

3. For each variable \( y \) construct an STG \( G^y \) from its state graph \( G^y \) and fragments \( G^{d^y} \); in each such STG transitions are labelled with the changes of variables in \( X^y \).

4. Construct a binary STG \( G^* \) from the implementable original SSTG \( G \), by substituting the fragments \( G^{d^y} \) for each symbolic transition in the original SSTG.

5. Check the implementability of STG \( G^* \).

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.

Before we perform these steps for our example we need to derive the sufficient conditions for the implementability of STG \( G^* \).

4 Implementability of Binary STG Expansion

4.1 Binary Expansion of Component Cliches

For every variable \( y \in Y \), whose local behaviour is defined by state graph \( G^y = (S^y, D^y) \), 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 \( \lambda \), consisting of two parts, \( \lambda^S \) and \( \lambda^D \), as follows:

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

\[
(\lambda_1 - \lambda_2)_j = \begin{cases}
+ & \text{if } \lambda_1 = 0 \land \lambda_2 = 1 \\
- & \text{if } \lambda_1 = 1 \land \lambda_2 = 0 \\
\ast & \text{if } \lambda_1 = \lambda_2
\end{cases}
\]

in which \( \lambda_1 = \lambda^S(s_2) \) and \( \lambda_2 = \lambda^S(s_2) \).
Therefore $\lambda^D$ 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 needs defining, for each $d$, a partial order on 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 possible as some intermediate states occurring between the main states in $S^y$ may coincide by their encodings. This may result in violation of the global USC condition, and hence the implementability of the final STG.

For every poset built on sets $\lambda^D(d)$ we can construct a (fragment of) $\text{STG}_{G^d}$, 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 $\Delta$) transitions, between which the STG $G^d$ is inserted as shown in Figure 2.

Avoiding unnecessary formalism, we state that by refining, for all $y \in Y$, each arc $d$ of $G^y$ in the above way we can always build a corresponding STG $G^y$. 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 $G$: 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 $G$.

**Property 4.1**

1. The underlying PN of the STG built for each variable cliche is a safe free-choice PN\(^3\).
2. The STG built for each variable's cliche is consistent.

This property implies that each local STG is valid.

### 4.2 Construction of Global Signal Transition Graph

The above refinement of transitions of variables in $Y$ can also be used in binary expansion of the original SSTG $G = (P, Y, \Delta)$ into an STG $G^* = (P^*, X, \Delta^*)$, where $X$ is the union-set

\(^3\)The definitions of free-choice, safe etc. Petri Nets can be found for example in [11].
Figure 3: Transformation of Symbolic Transition Graph into Global Signal Transition Graph

of all encoding variable sets $X^v$. Thus each labelled transition $t \in T$ in the underlying PN $P = (P, T, F, m_0)$ we substitute with a fragmental STG $G^d$ as shown in Figure 3.

The following proposition holds.

**Proposition 4.1** The global STG expansion of a valid SSTG is also valid iff all the variables’ STG expansions are valid.

### 4.3 Implementability Conditions

#### 4.3.1 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 $S^y$ 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, $S^y$ contains all the states that $G^y$ has (encoded by the corresponding $\lambda^S$), 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.

**Theorem 4.2** 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 theorem is important since it helps to reduce the problem of checking the global STG’s implementability to a number of subproblems of much smaller dimension, 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 severely constrain
concurrency of the transitions of the encoding variables from $X^y$. It is intuitively clear that
some local USC violations can be "compensated" by potential 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 distinguishing variables we shall only consider those whose values
in the states "plagued" by USC violations are stable, so we can fully benefit from remaining at
the symbolic level.

4.3.2 Weak Implementability

Assume Strong Implementability does not hold, and a variable $y \in Y$ whose STG has USC
problem, i.e. its associated STD $S^y = (\Sigma^y, E^y, \delta^y)$ is contradictory. Therefore, there is at least
one pair of states $\sigma_1$ and $\sigma_2$ in $\Sigma^y$ such that $\delta^y(\sigma_1) = \delta^y(\sigma_2)$. Any such pair of states are called
locally indistinguishable. It is obvious that for every nodal state $\sigma$ in $S^y$, $\delta^y$ is related to the
encoding function $\lambda^{S^y}$, 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 $\sigma_1$ and $\sigma_2$ cannot be nodal since $\lambda^{S^y}$ is injective by construction.
Therefore, we have two remaining cases:

1. $\sigma_1$ is transient and $\sigma_2$ is nodal;

2. both $\sigma_1$ and $\sigma_2$ are transient.

Consider only the first case (the case with transient $\sigma_2$ and nodal $\sigma_1$ is symmetrical). The second
one can be treated in the same way.

It is easy to see that, by our construction of the local STG the generated STD $S^y$ has a unique
nodal state, $\sigma_1$, which is the nearest predecessor of $\sigma_1$. $\sigma_1$ occurs in the transition of $y$ from
the state corresponding to $\sigma_1$ to some other nodal state. By the above relationship between $\delta^y$ and $\lambda^{S^y}$ for nodal states, we can uniquely derive the values of $y$, $v_1 = (\lambda^{S^y})^{-1}(\delta^y(\sigma_1))$ and $v_2 = (\lambda^{S^y})^{-1}(\delta^y(\sigma_2))$.

Let $S_1 \subseteq S$ ($S_2 \subseteq S$) be the set of states of the ALTS $S = (S, E, \delta)$ generated by the original SSTG
$G$ such that in all states in $S_1(S_2)$ component $y$ has value $v_1$ ($v_2$). It is clear that $S_1 \cap S_2 = \emptyset$.
Let $s_1 \in S_1, s_2 \in S_2$. 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 $\text{diff}(s_1 \setminus y, s_2 \setminus y)$ denote the set of variables in $Y \setminus \{y\}$
whose values are stable and different in their labellings $\delta(s_1 \setminus y)$ and $\delta(s_2 \setminus y)$.

Now, for values $v_1$ and $v_2$ of $y$ we construct a characteristic Boolean function, $F(v_1, v_2)$, called
Similarity Function, such that $F(v_1, v_2) = 1$ if there exists at least one pair $s_1 \in S_1, s_2 \in S_2$
such that $\text{diff}(s_1 \setminus y, s_2 \setminus y) = \emptyset$, otherwise $F(v_1, v_2) = 0$.

Our nodal states $\sigma_1$ and $\sigma_2$ are called indirectly distinguishable if $F(v_1, v_2) = 0$ for $v_1$ and $v_2$
obtained from $\sigma_1$ and $\sigma_2$ in the above way.

Since any transient state $\sigma \in \Sigma$ has a unique nearest nodal state $\bar{\sigma}$ and $\text{diff}(s_1 \setminus y, s_2 \setminus y)$ is
preserved in any firing sequence where $y$ changes between $\bar{\sigma}$ and $\sigma$, the above definition can be
extended onto any pair of states in $\Sigma^y$.

A pair of states $\sigma_1$ and $\sigma_2$ in STD $S^y$ generated by STG $G^y$ are called indirectly distinguishable
if their nearest predecessor nodal states are indirectly distinguishable.
With the aid of the above argument we can state another, less restrictive, implementability condition.

**Theorem 4.3** 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.

This condition is called *Weak Implementability*. Unfortunately, even this condition is only sufficient and could be weakened even more. However, any such weakening would involve a much more thorough analysis of distinguishability, e.g. by modifying Similarity 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.

We should point out that the above approach suggests two useful strategies for a constrained optimisation procedure:

- 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 concurrency constraints at the local cliche level, by allowing up to maximum concurrency between transitions of the encoding variables in \( \lambda^{D_y} \) for all y, until Weak Implementability no longer holds.

- Begin with the maximally concurrent local STGs, and while incrementally introducing more ordering constraints in \( \lambda^{D_y} \) for all y, check when Weak Implementability becomes true.

**4.4 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 \( G^* \) (see Figure 1.(b)) is shown in Figure 4.(a).

Here, \( \lambda^D(r^{0-1}) = \{r_1+, r_2+\} \). 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 \( G^* \) for r is shown in Figure 4.(b). The global binary STG \( G^* \) is shown in Figure 4.(c). It is easily seen that this STG is implementable. Using one of the existing methods [12, 4, 8] for ordinary STG we obtain the following set of Boolean functions:

\[
\begin{align*}
    b &= r_1 + r_2 \\
    m &= r_1 + r_2 \\
    r_1 &= bm + (\bar{m} + \bar{r}_2)r_1 \\
    r_2 &= \bar{b}r_1 + r_1r_2
\end{align*}
\]

The circuit implementing the interface controller component in Figure 1.(a) is shown in Figure 4.(d). The appropriate “request/acknowledgement” pairs of handshake signals are formed by breaking the interconnections of signals b, \( r_1 \) and m.

If we try to avoid serialising \( r_1- \) and \( r_2- \), then the local STG with concurrent transitions \( r_1- \) and \( r_2- \) will not be implementable, since the STD generated by it will contain two states, both
Figure 4: Interface Example: Synthesis
labelled with 10. One, nodal state, is reachable from the initial state, 00, by the firing of \( r_1 + \). The other, transient, 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 011 and 021, respectively, in Figure 1.(d) cannot distinguish the above two 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 5.(a). 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 5.(b), does no longer preclude implementation of the global binary STG expansion, shown in Figure 5.(c). The latter produces the following logical functions:

\[
\begin{align*}
b &= m + \bar{r}_1 + r_2 \\
m &= mr_1 + r_2 \\
r_1 &= b\bar{m} + m + r_1 \\
r_2 &= b\bar{m}r_1 + \bar{m}r_2
\end{align*}
\]

5 Design 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 using a packet switching/routing architecture. We assume that the channel has a data flow structure which allows data to be transferred in two directions in a time-shared way. The structural description is shown in Figure 6. Here, three adjacent cells, \((i-1)\)-th, \(i\)-th and \((i+1)\)-th, are extracted for convenience. When the pipeline is in the left-to-right ("\( \rightarrow \)"or "lr") mode a datum is stored 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 ("\( \leftarrow \)" or "rl") mode a datum is stored from the right neighbour cell, which is controlled by another strobe, \( Str_{rl} \).

Our goal is to derive a logical implementation for the controller of the \( i \)-th cell, which will generate the strobe signals depending on the direction mode that is set in the controller during special "mode change sessions" in the pipeline operation. Since we assume that the whole pipeline operates in a totally distributed way, the information about a particular mode is also communicated between the cells through the same pipeline. To facilitate this we arrange that
Figure 6: Two-way Pipeline: Structural View

Figure 7: Basic Asynchronous Control Structure for Controller Cell

once data is being transmitted from one terminator cell only this terminator owns the privilege to transmit data. The same terminator may stop sending data and transmit the privilege to the other terminator. The latter, after using the channel in similar way, may return the privilege back to the first terminator and the process repeats.

Figure 7 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 behavioural cliches of all variables in \( Y \) should be of the same type (pipeline uniformity), we present a generic state graph for just one variable, say \( y_i \). It is shown in Figure 8.(a). The variable has four states \( S = \{ \bar{S}, \bar{D}, S, D \} \):

1. \( \bar{S} \) stands for the spacer in the \( \rightarrow \) mode;
2. \( \bar{D} \) stands for the data valid state in the \( \rightarrow \) mode;
3. \( \bar{S} \) stands for the spacer in the \( \leftarrow \) mode;
4. \( \bar{D} \) stands for the data valid state in the \( \leftarrow \) mode;
Figure 8: First idea: State Graph (a), SSTG (b) and ALTS (c)
Corresponding notation is used to model the allowed changes of values of the variable: \( D^v = \{ \wedge, \vee, \backslash, /, \langle, \rangle, >, < \} \).

We specify the process in which the three adjacent cells interact by the SSTG shown in Figure 8.(b). This SSTG is built on a free-choice PN. 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 manifest the state in which all the cells are in the \( \overline{S} \), 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 onto \( \overline{S} \). If the first option is chosen the leftmost cell changes to \( \overline{D} \), and the whole pipeline behaves like an ordinary one-way pipeline, again until the point where the leftmost cell, while being in \( \overline{S} \), decides to hand the privilege by changing to \( S \). The \( i \)-th cell, when it accepts \( \overline{S} \) from the left-hand neighbour also changes from \( \overline{S} \) to \( S \). When the privilege reaches the rightmost cell, the latter begins either transmitting data or returning the privilege back to the left-hand neighbour. It is easy to see that the whole behaviour is similar to the above, except for the roles of “right” \( (i+1) \) and “left” \( (i-1) \) have been interchanged.

One can see that the SSTG is valid (its PN is bounded and it is consistent), however it appears to have a USC problem, which is seen from the ALTS generated by the SSTG and shown in Figure 8.(c). This ALTS is contradictory and the pairs of different states (linked by dotted lines), labelled with the same combinations of values of variables, indicate that the output variable \( y_i \) is unable to determine from the state of the other two variables, whether to remain stable (state \( \overline{S}, \overline{S}, \overline{S}^* \)) or become excited (state \( \overline{S}, \overline{S}, \overline{S} \)). This problem results in the fact that even our SSTG is not implementable, regardless of the implementability of the local behaviour of variables.

5.2 Second Idea: Correct Solution

One can easily notice from the ALTS generated by the previous SSTG that the major problem hides 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 introducing a pair of special states (for either directions), \( \overline{C} \) and \( \overline{C} \) into the local behavioural cliches of the variables in \( Y \), which explicitly manifest the change of the transmission mode. Such a state can be interpreted as a special value, called “privilege”, transmitted through the channel. The cliche’s state graph is shown in Figure 9.(a). With the pair of new states we have also obtained the new changes, \( >1, >2, <1 \) and \( <2 \), whose meaning is quite obvious.

The SSTG satisfying the above cliche is shown in Figure 9.(b). This graph has no USC problem and, since being valid, is implementable (the reader may try to built its ALTS as an exercise).

We proceed further with the binary encoded state graph shown in Figure 9.(c). The set of encoding variables is \( X^v = \{ a_i, b_i, c_i \} \). This graph gives rise to the local STG, shown in Figure 9.(d), which is in this case built uniquely 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 built trivially from the SSTG and the encoded transitions of variables. For example, \( (i-1) \) \( \langle \) is substituted with the transition labelled with \( a_{i-1}+ \), \( (i-1) \rangle \) with \( b_{i-1}+ \) and so on.

Since the local cliches are also implementable, Theorem 4.2 implies that the global STG is implementable too. Using a standard technique [12, 4, 8] we obtain the logical implementation.
Figure 9: Correct Solution: State Graph (a), SSTG (b), Encoded State Graph (c) and local STG(d)
for output signals, \( a_i, b_i \) and \( c_i \):
\[
\begin{align*}
a_i &= a_{i-1} \bar{a}_{i+1} c_i + a_{i-1} a_{i+1} c_i + a_i (\bar{a}_{i-1} c_i + a_{i-1} \bar{c}_i + a_{i+1} c_i + \bar{a}_{i+1} \bar{c}_i) \\
b_i &= a_{i-1} c_i + \bar{a}_{i+1} b_{i-1} c_i + b_{i+1} \\
c_i &= b_{i-1} c_i + b_{i+1} c_{i-1} + c_{i+1}
\end{align*}
\]

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 \( Str_{rl} = a_i b_i \).

The actual interconnection refining the abstract ACS in Figure 7 is shown in Figure 10. 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 \( a_{\text{req}} \). This signal is used to strobe the AND gates for \( Str_{lr} \) and \( Str_{rl} \). The acknowledging input signal from the data path should be connected to the other end of the above mentioned broken wire (\( a_{\text{ack}} \)), and then can be used for both internal and external interconnections of the \( i \)-th cell. It is easily seen from the local STG that signal \( b_i \) can be used for the AND elements directly because its state is always stable when the circuit changes the value of \( a_i \).

### 5.3 Third idea: Restrictive Solution

Analysis of the local cliche in Figure 8.(a) shows that the required distinguishing effect could also have been achieved without introducing extra states, and, hence, extra encoding signals. The transfer of the privilege from one terminator to the other can be manifested not by the transition from one spacer to the other but rather by resetting, after the final datum has been transmitted, directly to the other spacer state. Such a behaviour is shown in Figure 11.(a).

Using this idea we can build the SSTG shown in Figure 11.(b). Simple check shows that it is implementable, so are the STG's of the individual variables, whose generic form is shown in Figure 11.(c). Here the encoding requires only two auxiliary signals, \( a \) and \( b \). After quite trivial substitutions, one can construct the global binary STG which is also implementable by Theorem 4.2.

The following logical functions have been obtained from it in a standard way:
\[
\begin{align*}
a_i &= a_{i-1} \bar{a}_{i+1} b_i + \bar{a}_{i-1} a_{i+1} b_i + a_i (a_{i-1} \bar{b}_i + a_{i-1} b_i + a_{i+1} b_{i-1} + \bar{a}_{i+1} \bar{b}_i) \\
b_i &= b_{i+1} + a_{i+1} b_{i-1} + b_i a_{i-1}
\end{align*}
\]

The functions for the output strobes can now be produced from the encoding of the "data valid" states: \( Str_{lr} = a_i \bar{b}_i \) and \( Str_{rl} = \bar{a}_i b_i \).
Figure 11: Restrictive Solution: State Graph (a), SSTG (b) and Local STG (c)
Unlike the previous solution allowing to acknowledge the fact of setting and resetting data in the multiplexed data path register (see Figure 10), this solution does not allow to organise fully delay insensitive cooperation between control and data path. (At least the authors have not been able to find a correct circuit.) However, the effect of signal *Ack*, generated by the data path, can be “simulated” by inserting adequate delays into the wires of $a_i$ and $b_i$ that are fed back to the control circuit, similar to the way the signal $a_i$ has been used in Figure 10. Such compromising decision can be justified if the data path of the $i$-th cell is physically implemented within the same region on the chip, and the delays in the interconnections between adjacent cells are assumed larger than those within the same cell.

This solution thus requires less circuitry and produces faster operation (no need to pass through a spacer state when changing the direction of data transfer). At the same time it is less flexible since the terminating cells are unable to withhold from data transmission by “bouncing” the privilege to each other, which was possible for the previous solution. To do this, one has to organise the transmission of a special dummy (at the higher communication level) datum that would hand the privilege to the other end.

6 Conclusions

With the proposed synthesis approach, based on a high-level specification language, Symbolic Signal Transition Graph, we have achieved the following two main results:

- Better way of capturing the system’s abstraction and hierarchy of its formal description.
- More efficient checking of the specification’s implementability conditions.

The circuit designer would gain maximum benefit of this approach if he would be able to describe the initial idea of a synthesised Asynchronous Control Structure in an essentially abstract form. The number of components should be relatively small, while their behaviour can be specified as a multi-state (not necessarily two-state like in “ordinary” Signal Transition Graph) graph, which is further refinable by binary encoding.

The sufficient conditions of the implementability of a binary expansion of the global specification suggest flexible optimisation strategies.

It would be interesting to combine this approach with the recent synthesis method, proposed by Chu [3], that is based on transforming an Asynchronous Finite State Machine specification into a Signal Transition Graph implementable by the existing techniques.

Acknowledgements: The authors would like to thank: Dr. Leo Rosenblum for the valuable input to this work; Luciano Lavagno for discussions on asynchronous circuit synthesis, the idea of the term SSTG and his advice on the SIS CAD tools, which have been used to derive some logical implementations; Newcastle University’s Department of Computing Science for helping to arrange the authors’ joint work in England. Helsinki University of Technology and Swedish Institute of Technology at Helsinki are also acknowledged for supporting the work of A. Petrov.
References


