Synthesis of Speed Independent Circuits from STG-unfolding Segment*

Alex Semenov, Alexandre Yakovlev
Department of Computing Science
University of Newcastle
Newcastle upon Tyne, NE1 7RU
England

Enric Pastor, Marco A. Peña,
Jordi Cortadella
Department of Computer Architecture
Universitat Politèchnica de Catalunya
08071 Barcelona, Spain

Abstract

This paper presents a novel technique for synthesis of speed-independent circuits. It is based on partial order representation of the state graph called STG-unfolding segment. The new method uses approximation technique to speed up the synthesis process. The method is illustrated on the basic implementation architecture. Experimental results demonstrating its efficiency are presented and discussed.

1 Introduction

The problem of synthesis of speed-independent circuits from their Signal Transition Graph (STG) specifications has been approached by many researchers. Several tools exist today, such as SIS [12], Assassin [15], Forcage [3] and Petrify [2], which are capable of synthesising circuits of moderate size. With the exception of Forcage, all tools use some form of State Graph (SG) representation to obtain truth tables of the implementation logic. Petrify uses Binary Decision Diagrams (BDDs) to represent SG symbolically and can thus synthesise circuits from larger descriptions. Forcage, on the other hand, uses Change Diagrams (partial order model) to derive an implementation but is restricted to specifications without choice.

Construction of SG hits available computational limits due to state explosion. A structural method in [8] can implement STGs avoiding exhaustive state exploration. It uses concurrency relation between transitions of the STG to obtain an initial approximation of the implementation. If this approximation does not satisfy correctness criteria, then iterative refinement is performed, where the procedure uses State Machine (SM) decompositions. Although powerful, this method has a drawback, it is restricted to free-choice specifications.

The main goal of this work is to develop a method for implementing STGs that cannot be synthesised by the above techniques due to the large size of their SG. A way to achieve this goal will be analogous to the one in [8] – it will draw upon relations at the event-based, rather

*This work was supported in part by the SERC grant No. GR/J 52327 and ESPRIT ACiD–WG Nr.21949. Collaboration between University of Newcastle and Universitat Politèchnica de Catalunya was supported by British-Spanish joint research programme (Acciones Integradas) between the British Council and Spanish Ministry of Education and Culture, grant number MDR(1996/97)1159

than state-based, description level. This method will, however, be free from the limitations of [8].

The solution to this problem is found in the use of a partial order approach, already known to have given positive results in STG verification. It is based on an implicit representation of SG in the form of a finite STG-unfolding segment [11]. It was shown [11] that such a segment can often be built for those examples where the construction of SG fails. While the segment is being constructed it is also verified for correctness. Thus, after the verification stage is completed, an implementation can be derived from an already built STG-unfolding segment. Two approaches are possible within the new synthesis method: exact and approximate. The former obtains an implementation equivalent to that derived from the SG. At the end of the synthesis procedure this approach produces an implementation by recovering binary states from the segment (similar to the approach of [6]). Although it benefits from the unfolding methodology which restricts the set of states needed to examine for each signal, the exact approach may suffer from exponential explosion of states. To battle the complexity, the latter approach uses concurrency relation to initially approximate and then to refine an approximated implementation. The structural method of [8] works on the STG level, assuming that two transitions are concurrent if they can ever fire simultaneously. On the contrary, our method works with a partial run of the STG specified behaviour. Thus it is possible to pin-point when exactly any two transitions become concurrent. This local information gives a more accurate initial approximation and a more precise refinement.

The aim of this paper is to suggest and illustrate synthesis of speed-independent circuits from the STG-unfolding segment built for their specifications. The method is illustrated on the atomic complex gate per signal architecture. The paper is therefore organised as follows. Section 2 describes general approach to synthesis of speed-independent circuits and describes the atomic complex gate per signal architecture. Section 3 introduces new notions required for the synthesis from STG-unfolding segment. Section 4 describes and illustrates the suggested method. Results showing performance and comparing the new method with existing ones are presented in Section 5.

2 Synthesis of Speed-Independent circuits

2.1 General synthesis approach

We assume that the reader is familiar with the basics of the Petri net theory [9]. A *marked Petri net (PN)* is a tuple $N = \langle P, T, F, m_0 \rangle$ where P and T are non-empty sets of places and transitions, respectively, F is a flow relation and m_0 is an initial marking. A *Signal (Transition) Graph* (STG) [10, 1] is a tuple $G = \langle N, A, L \rangle$ (labelled PN) where N is a marked PN, A is a set of signals and $L : T \to \{+, -\} \times A$ is a labelling function. STGs are a special case of labelled PNs, used for low level descriptions of asynchronous circuits. The set of transition labels represents changes of signals: $+a_i$ (for up) and $-a_i$ (for down). Notation $*a_i$ indicates a transition labelled with a change of a_i regardless of the direction of this change.

Conventionally, to obtain an implementation for an STG G a corresponding SG is built. The SG S, also called *State Transition Diagram* (STD), is derived by constructing the reachability graph (representing all reachable markings) of the underlying PN and then assigning binary codes v_i to each vertex s. The binary codes must be assigned *consistently*, i.e. :

- every arc is labelled with exactly one signal transition, and
- for each pair of states s_1 and s_2 connected with an arc labelled with $*a_i$ the following is true:
  - $v_1[i] = 0$ and $v_2[i] = 1$ if $* = +$
  - $v_1[i] = 1$ and $v_2[i] = 0$ if $* = -$

Once a consistent state assignment was performed, truth tables are obtained for each output signal and an implementation is produced. The process of obtaining a truth table depends on the implementation architecture chosen (for this particular signal).

Correctness criteria for synthesis of speed-independent circuits can be divided into *general correctness criteria* and *architecture specific correctness criteria.* The former are behavioural properties of an STG, which characterise an STG to be implementable "in principle". In addition to the consistent state assignment, they also include:

- *Boundedness*, which guarantees that the behaviour specified by an STG can be implemented into a finite size circuit;
- *Semi-modularity* (also called "output signal persistency"), which implies that excited output signals cannot be disabled by some input signal change and thus cause a hazard.

The latter group of properties is usually checked during the actual logic synthesis process. These are generally referred to as *coding conflicts* and indicate that although the STG is implementable "in principle", there may exist some binary state associated with different markings which makes them indistinguishable at the circuit level. The *Complete State Coding* (CSC) condition introduced in [1] requires any two states with equal binary codes to have the same set of excited output signals. It was proved in [1] that STGs satisfying CSC property are implementable as speed-independent circuits.

An implementation is obtained by building a cover function. A boolean function with a variable corresponding to each signal is said to be *covering* a state s_j if it evaluates to TRUE when the variables have the values equal to the elements of binary code v_j assigned to s_j. A function $\mathcal{C}$ covering a set of states is called a *cover function* (or simply **cover**) for this set of states $\{s_i\}$. Each term of the cover is called *cube* as it may cover several states in the state space.

A cover is not required to be *exact*, i.e. to cover *only* the states in $\{s_i\}$. It could be obtained explicitly from their binary codes. However, if a cover is obtained somehow differently (e.g. using an oracle), it may cover some other states. For example, methods described in [14, 8] use structural information to obtain covers. Such cover is called *approximated cover*, and needs to be checked for correctness. There are different requirements for correctness of covers according to the implementation architecture chosen.

The following three architecture types are normally considered:

- *Atomic complex gate per signal* implementation;
- *Atomic complex gate per excitation function* implementation;
- *Atomic complex gate per excitation region* implementation.

The first architecture can be considered as a basic type. The other two aim at reducing the size of customised complex gates. In these architectures it is assumed that the output signal is implemented using a memory element. The Set and Reset *excitation* functions for this memory element are implemented as atomic complex gates (the former) or a network of atomic complex gates (the latter). Depending on which memory element is used, the implementations are divided into **i)** *Standard C-element* implementation, which uses Muller C-element as the memory element, and **ii)** *RS-latch* implementation, where an RS-latch is used.

To demonstrate the novel technique we chose the atomic complex gate per signal architecture. Our method can be easily adapted to the other architectures. The remainder of this section overviews the synthesis strategy for the chosen architecture type. Note that the limits of this paper also do not allow us to consider lower level logic decomposition, usually known as technology mapping.

2.2 Atomic complex gate per signal implementation

This is a basic architecture for speed-independent circuits studied in [1, 5]. The circuit is implemented as a network of atomic gates. Each gate uniquely implements one output signal. Its boolean function can be represented as Sum-Of-Products (SOP) or Sum-Of-Functions(SOF). An example of such gate is shown in Figure 1. Each gate is allowed to be sequential (latch), i.e. contain an internal feedback with a zero delay. The delay between its internal "ANDing" and "ORing" parts is also assumed to be negligible. The gate depiction is used to denote the implemented boolean function as the actual implementation is resolved on the transistor level.

Two sets of the reachable states are distinguished in the SG, *on-set* $On(a_i)$ and *off-set* $Off(a_i)$, which include all states in which the value of the output signal a_i is implied to be TRUE and FALSE, respectively. The remaining (unreachable) subset of combinations of the boolean values of signals forms the *Don't care* set (DC-set).

The implementation is derived by building the on-set[1]. Each state can be represented by a term which has $|A|$ variables, each corresponding to one and only one signal a_i. The term becomes TRUE only when the values of the variables are equal to those in the binary code assigned to the state. The cover $\mathcal{C}$ for implementation is obtained from the terms included into the on-set. The DC-set can be used for optimising the size of $\mathcal{C}$. This is done in standard minimisation tools, such as Espresso [12].

The synthesis for this architecture is illustrated in Figure 1 for an STG shown in the Figure 1(b). Suppose that an implementation of the signal b is required. The on-set of b is found as: $On(b) = \{(p_2, p_3), (p_3, p_5), (p_2, p_6, p_8), (p_5, p_6, p_8), (p_7, p_8), (p_4)\}$. The cover function $\mathcal{C}(b)$ is obtained as: $\mathcal{C}(b) = a\bar{b}\bar{c} + ab\bar{c} + a\bar{b}c + abc + \bar{a}bc + \bar{a}\bar{b}c = a + c$. The DC-set in example in Figure 1(c) is empty so no further minimisation can be done.

Obtaining exact covers usually means that all states in the on- or off-set must be known. An approximation algorithm produces approximated covers of the on- and off-sets. Therefore, in this implementation architecture, covers of on- and off-sets must satisfy the following condition:

Definition 2.1 Two covers $\mathcal{C}^*_{On}(a_i)$ and $\mathcal{C}^*_{Off}(a_i)$ are said to be correct iff $\mathcal{C}^*_{On}(a_i)$ and

[1]Here and further, for simplicity, it is assumed that the on-set is constructed. Usually, the simplest from the on- and off-sets is chosen for implementation.

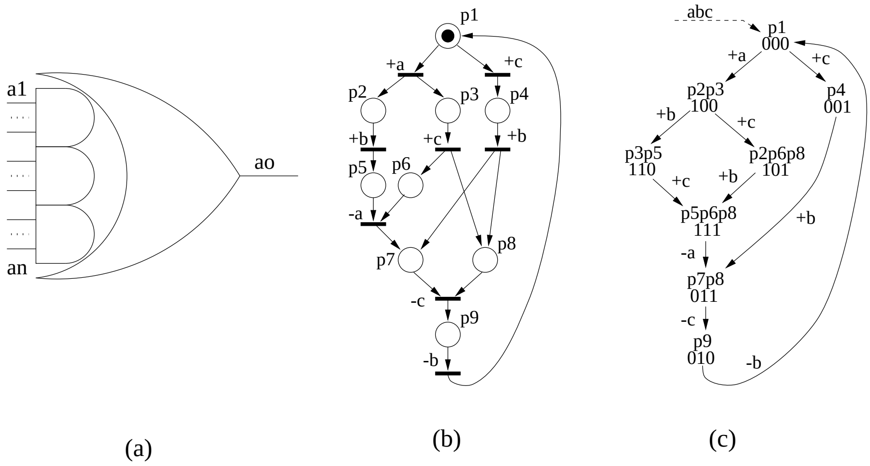


Figure 1: An example of an STG and a corresponding SG.

$\mathcal{C}^*_{Off}(a_i)$ cover $On(a_i)$ and $Off(a_i)$ respectively and $\mathcal{C}^*_{On}(a_i) \cdot \mathcal{C}^*_{Off}(a_i) \subseteq$ DC-set. $\square$

If the covers do not satisfy the above condition, then the approximation is too loose and needs to be refined. If, on the other hand, the covers are exact but still intersect outside the DC-set, then this STG has CSC problem. In this case it should be corrected by changing the specification, e.g. by inserting additional signals.

3 Slices in STG-unfolding segment

3.1 STG-unfolding segment

Analysis of STGs using STG-unfolding segment was studied elsewhere [11]. An STG-unfolding segment is a tuple $G' = \langle T', P', F', L' \rangle$ where T', P' and F' are sets of transitions, places and the flow relation, respectively, and L' is a labelling function which labels each element of G' as an instance of elements of G. G' is a partial order obtained from an STG G by the process of its unfolding which starts from the initial marking. The unfolding process uses the structural properties of the constructed partial order to determine the relations of *conflict*, *concurrency* and *precedence* between instances. These relations are used to decide where to instantiate the next element. The following key notions were introduced in [4]:

- The min-set of transitions needed to fire t', including t', is called *local configuration* of t' and is denoted as $\lceil t' \rceil$.
- A set of place instances reached by firing all transitions in $\lceil t' \rceil$ is called *postset of* $\lceil t' \rceil$ and is denoted as $\lceil t' \rceil\bullet$. Mapping a postset onto places of the original STG is called *final state of* $\lceil t' \rceil$ and gives a marking of the original STG.
- Any non-conflicting and transitively closed set of transitions of T' is called *configuration* C. The postset of a configuration, denoted as $C\bullet$, is found from the postsets of transitions comprising it.

The unfolding algorithm examines only states reached through firing of an instance t' excited by a minimal set of causes. It is based on the fact that no new information about the behaviour of the system can be obtained once the states started repeating. Thus the algorithm constructs no new instances after any instance t'_c whose firing reaches an already examined state. Transition instance t'_c is called a *cutoff transition* of the unfolding.

In contrast to PN-unfolding [4], the STG-unfolding takes into account specific signal interpretation of PN transitions and keeps track of the binary codes reached by transition firing. However, it still examines only a subset of all reachable states of STG G and thus is more efficient than SG analysis for a vast number of examples.

Each instance t' of STG-unfolding segment is assigned with a *binary code* $\xi_{\lceil t' \rceil}$ which is reached by firing transitions in $\lceil t' \rceil$. Similar to its postset, the binary code corresponding to a configuration C is calculated from $\xi_{\lceil t' \rceil}$ of transitions comprising it. It was shown in [11] that all states of the SG are represented in the STG-unfolding segment as postsets of some configuration. For each instance t' labelled with signal transition $*a_i$ a set of transitions $next(t')$ is defined as a set of instances labelled with $*a_i$ reachable from t' without any intermediate transitions of a_i. Set $first(a_i)$ is a set of transitions of a_i first reached from the beginning of the segment. A special transition, called *initial transition*, is introduced in the unfolding to represent the initial state of the STG. This transition, denoted as $\perp$, has a postset which maps onto the initial marking m_0 and has an assigned binary code $\xi_{\lceil \perp \rceil}$ equal to the initial binary state v_0 of the STG.

It was demonstrated in [11] that an STG-unfolding segment can only be constructed for an STG specification satisfying boundedness and consistent state assignment criteria. The last general correctness criterion, semi-modularity, can be checked on the STG-unfolding segment in linear time.

3.2 Cuts

To represent a state of SG we define a cut. *A cut of* STG-*unfolding segment* is a maximal set of concurrent places $p' \in P'$. Each cut c of an STG-unfolding segment thus represents some reachable marking of the original STG. A sequence relation is defined between two cuts $\mathsf{c}_1 \preceq \mathsf{c}_2$ if $\forall p'_i \in \mathsf{c}_2, \exists p'_j \in \mathsf{c}_1 : p'_j \preceq p'_i$. For each instance t' the following four types of cuts are found.

- A *minimal excitation cut* $\mathsf{c}_e^{min}(t')$, which represents a state at which t' becomes first enabled.

- A *minimal stable cut* $\mathsf{c}_s^{min}(t')$, which represents a state which is reached by firing of t'.

- A *maximal excitation cut* $\mathsf{c}_e^{max}(t')$, which represents a state from which, in a correct STG no advancement can be made unless t' is fired.

- A *maximal stable cut* $\mathsf{c}_s^{max}(t')$, which represents a state which is reached after firing of t' from which firing of any transition leads to a state enabling the next change of the signal a_i labelling t'.

Each instance of the STG-unfolding segment uniquely identifies $\mathsf{c}_e^{min}(*a'_i)$ and $\mathsf{c}_s^{min}(*a'_i)$ and the *sets* of $\mathsf{c}_e^{max}(*a'_i)$ and $\mathsf{c}_s^{max}(*a'_i)$. Thus each instance identifies states bounding the subset of the on-set (or off-set) of a_i which is found for this particular instance.

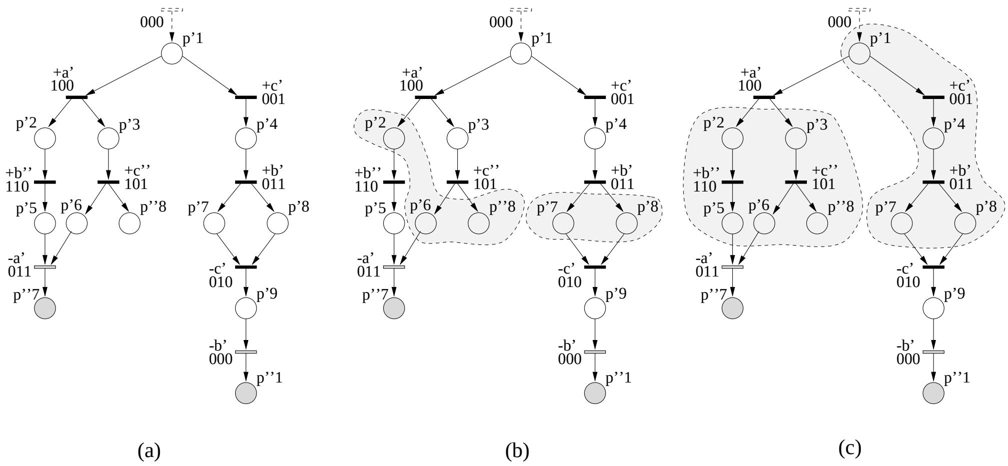


Figure 2: An example of an STG-unfolding segment and illustration of slices and cuts.

3.3 Slices

To represent a (connected) set of states we introduce a notion of a slice of the STG-unfolding segment. A *slice of* STG-*unfolding segment* is a set of cuts $\mathcal{S} = \langle \mathsf{c}^{min}, \mathsf{C}^{max} \rangle$ defined with a *min-cut of the slice*, c^{min}, and a *set of max-cuts*, C^{max}, such that $\forall \mathsf{c}_i \in \mathcal{S}$ the following is true: $\mathsf{c}^{min} \preceq \mathsf{c}_i$ and $\exists \mathsf{c}_j^{max} \in \mathsf{C}^{max} : \mathsf{c}_i \preceq \mathsf{c}_j^{max}$. No two cuts in the set of max-cuts are sequential.

In other words, a slice is defined between one min-cut and a set of max-cuts. Every cut in between the min-cut and a max-cut is encapsulated in the slice $\mathcal{S}$. Furthermore, for any two cuts c_i and c_j encapsulated by $\mathcal{S}$, if $\mathsf{c}_i \prec \mathsf{c}_j$, then all cuts between c_i and c_j are also encapsulated by $\mathcal{S}$. Since each cut represents some state in the SG, for any two states s_i and s_j represented as sequential cuts in a slice, all states on any path from s_i to s_j are also represented as cuts encapsulated into $\mathcal{S}$. The number of cuts in the set of max-cuts corresponds to the number of configurations (non-conflicting runs of the STG) which include configuration producing the min-cut. The elements of the STG-unfolding segment, i.e. places and transitions, bounded by instances in min-cut and max-cuts are said to belong to the slice.

A slice represents a subset of reachable states found in the SG for any STG bounded by the cuts defining it. As discussed earlier, the synthesis of speed-independent circuits is based on finding subsets of reachable states. Therefore, slices of the STG-unfolding segment can be used to identify and represent these subsets.

Cuts and slices are illustrated in Figure 2. Consider a cut $\mathsf{c} = (p'_7, p'_8)$ in Figure 2(b). This cut is a minimal excitation cut for the transition $-c'$ and is a minimal stable cut for $+b'$. Another cut, $\mathsf{c} = (p'_2, p'_6, p''_8)$ is a maximal stable cut for transition instance $+a'$. At the same time this is a maximal excitation cut for the instance $+b''$. This example also illustrates the relations between cuts. Intuitively, if a transition $*a'_i$ causes $*a'_j$, then the minimal stable cut of $*a'_i$ is the minimal excitation cut of $*a'_j$ and vice versa.

Slice $\mathcal{S}_1 = \langle (p'_1), \{(p'_7, p'_8)\} \rangle$ encapsulates cut(Figure 2(c)) $\mathsf{c} = (p'_4)$. Another slice $\mathcal{S}_2$ is defined between a min-cut (p'_2, p'_3) and a set of max-cuts $\{(p'_5, p'_6, p''_8)\}$ and includes all cuts between them. It is also possible to define a slice between (p'_2, p'_3) and $\{(p'_3, p'_5), (p'_2, p'_6, p''_8)\}$. In this case the slice will include all cuts but one enabling $-a'$. This slice, therefore, represents all states at which signal a is stable at "1".

Each cut is produced by some configuration of the STG-unfolding segment. Hence, the binary codes of the SG states represented by cuts encapsulated in a particular slice can be recovered by examining its cuts.

4 Synthesis from STG-unfolding segment

4.1 Obtaining exact covers

First, consider the problem of synthesis from the STG-unfolding segment G' by finding exact covers for the on-(off-)set. To implement an output signal of an STG as an atomic gate, its on-set[2] is required. Since its SG is represented as an STG-unfolding segment, the problem is to find a set of slices in this segment which represents all states in the on-set, i.e. an on-set partitioning of G' for a_i.

To define each slice we need to identify a min-cut and a set of max-cuts. From all instances in the STG-unfolding segment only instances of $+a_i$ may change the value of corresponding element in the binary codes. Furthermore, for each instance $+a_i'$ its minimal excitation cut $\mathsf{c}_e^{min}(+a_i')$ represents the first state at which $+a_i'$ becomes excited. Any cut at which $+a_i'$ is excited or stable at "1" must be sequential to $\mathsf{c}_e^{min}(+a_i')$. A special case is the initial transition $\perp$ of G'. If in the initial state of the STG the corresponding bit of binary code was "1", then the set $first(a_i)$ will consist of the down instance $-a_i'$. In this case, the minimal stable cut of $\perp$ is the first cut from which this slice can be defined. Thus the set of minimal cuts, which is used to define a set of slices, is taken as a set of minimal excitation cuts of instances $+a_i'$ and the minimal stable cut of $\perp$, if the signal a_i is at "1" in the initial state. Thus a set of transitions, called *entry transitions*, is identified on the STG-unfolding segment which includes all instances of $+a_i$ and may include $\perp$ if a_i is at "1" in the initial state.

For complete definition of each slice we need to determine a set of max-cuts for each slice. The minimal excitation cut of any instance $-a_i'$ represents the first state at which $-a_i'$ becomes excited. This cut belongs to the off-set.

For each instance $+a_i'$ the slice must be bounded by a set of cuts which can be reached from min-cut without exciting $-a_i$. The slice is bounded by the maximal excitation cuts of immediate predecessors of $next(+a_i')$, i.e. cuts at which an immediate predecessor of a transition from $next(+'a_i)$ is the only transition to fire. This is the furthest state to which advancement of the system can be made from $+a_i'$ without enabling $-a_i$. In the case of initial transition the set of max-cuts for the first slice is chosen using $first(a_i)$.

Due to the unfolding algorithm, a particular configuration may contain no instances of $-a_i$. This may happen if the configuration contains a cutoff transition, or simply leads to a deadlock. In this case the cut reached by such configuration bounds the slice.

Consider synthesising signal b from an example in Figure 1. The on-set partitioning of the segment is shown in Figure 3. There are two instances $+b'$ and $+b''$ and one instance $-b'$. Thus there are two slices $\mathcal{S}^1_{On}(+b') = \langle (p_4'), \{(p_7', p_8')\}\rangle$ and $\mathcal{S}^2_{On}(+b'') = \langle (p_2', p_3'), \{(p_5', p_6', p_8'')\}\rangle$ representing states from the on-set and one slice $\mathcal{S}_{Off} = \langle (p_9'), \{(p_1'')\}\rangle$. Once the slices are defined, the set of states represented by these slices is found: $On^1(b) = \{100, 101, 110, 111\}$ and $On^2(b) = \{001, 011\}$. The on-set cover is obtained from slices as $\mathcal{C}_{On} = On^1(b) \cup On^2(b) = \{100, 101, 110, 111, 001, 011\}$ which after standard boolean transformation gives $\mathcal{C}_{On} = \{1--, --1\} = a + c$. If the off-set implementation were chosen,

[2] Off-set if an off-set implementation was chosen. In this case instances of $-a_i$ should be considered.

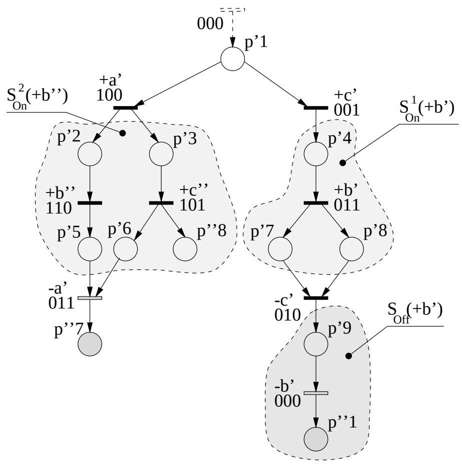


Figure 3: Illustration of synthesis from the STG-unfolding segment.

then the cover would be $\mathcal{C}_{Off} = \{010, 000\} = \bar{a}\bar{c}$.

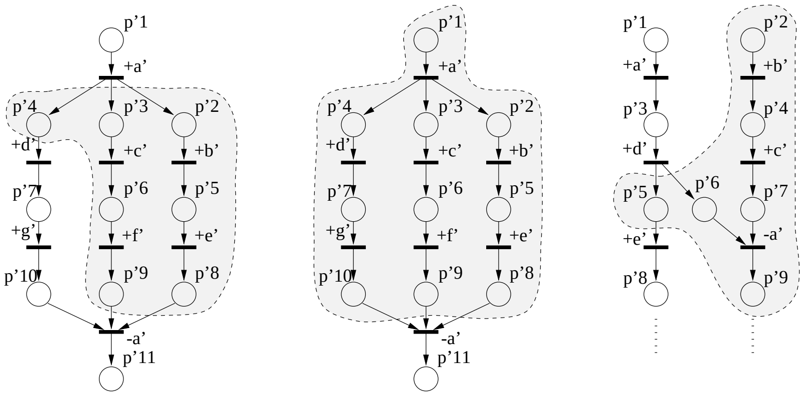
4.2 Deriving cover approximation from STG-unfolding segment

The synthesis procedure described in the previous Subsection suffers from one drawback. If many concurrent transitions belong to a slice, then obtaining the binary codes for all cuts will suffer from exponential explosion of states. To battle this an approximation method is suggested.

Two types of nodes can be identified in the on-set of signal a_i: those which have $+a_i$ excited and those at which a_i is stable at "1". The former is traditionally called *excitation region* (ER) and the latter *quescient region* (QR) of $+a_i$. A set of states at which a particular place p_l is marked is called a *marked region* (MR) of this place. It was pointed out in [7] that a cover for any set of states can be found as an intersection of covers for places which are marked at each state. Thus a set of states at which a particular transition is excited can be found as an intersection of MRs of its preceding places. However, at the unfolding level the instances of transitions are known. Furthermore, the minimal excitation cut $\mathsf{c}_e^{min}(*a_i')$ for each instance $*a_i'$ indicates where this instance becomes first enabled.

Any state reachable from $\mathsf{c}_e^{min}(*a_i')$, preserving the excitation of $*a_i'$, can only be reached by firing transitions which are concurrent to $*a_i'$. If a signal transition instance $*a_j'$ is concurrent to $*a_i'$, then the value of its corresponding element in the binary code may take values of both "0" and "1". A cover approximation $\mathcal{C}_e^*(a_i')$ is found from the binary code ξ assigned to the cut $\mathsf{c}_e^{min}(*a_i')$. Literals corresponding to signals whose instances belong to $\mathcal{S}_e(*a_i')$ and are concurrent to $*a_i'$ are substituted by "–" (don't care). Approximation reduces the number of literals in cover $\mathcal{C}_e^*(a_i')$ and increases the number of combinations covered by $\mathcal{C}_e^*(a_i')$. However, such approximation guarantees that no marking at which $*a_i'$ is excited was lost. Furthermore, for a CSC-compliant STG, $\mathcal{C}_e^*(a_i')$ will only cover those reachable states where $*a_i'$ is excited.

For example, consider calculation of $\mathcal{C}_e^*(+d')$ for the instance $+d'$ in Figure 4(a). The binary code corresponding to its minimal excitation cut $\mathsf{c}_e^{min}(+d') = (p_2', p_3', p_4')$ is found from the binary code of its local configuration $\lceil +d' \rceil$ as $\xi = \{1000000\}$ (the order of signals is $abcdefg$). There are four signals $\{b, c, e, f\}$ whose instances belong to the slice and are concurrent to $+d'$. Thus the ER cover approximation for $+d'$ will be $\mathcal{C}^*(+d') = \{1 - -0 -$



(a) (b) (c)

Figure 4: Illustration of cover approximation and refinement.

$-0\} = a\bar{d}\bar{g}$.

The rest of the states in the on-set which are represented as cuts encapsulated by $\mathcal{S}_{On}(*a'_i)$ can be approximated by taking cover approximations for MRs of places belonging to $\mathcal{S}_{On}(*a'_i)$ and sequential to the entry transition of the slice.

For each place p'_l its MR approximation cover $\mathcal{C}^*_{mr}(p'_l)$ is obtained from the binary code $\xi_{\lceil t'_k \rceil}$ assigned to its preceding transition. Similar to ER approximation, any marking at which p'_l is marked can only be reached by firing transitions concurrent to p'_l. Thus literals corresponding to signals whose instances belong to $\mathcal{S}_{On}(*a'_i)$ and are concurrent to p'_l are replaced by "–".

An MR cover approximation for a particular place p'_l will cover all states at which p'_l is marked with any other concurrent place p'_j. Thus only mutually non-concurrent subset of places belonging to $\mathcal{S}_{On}(*a'_i)$ can be considered. A set of such places is called *approximation set* P'_a. Furthermore, an MR cover approximation must not cover markings enabling instances $t'_j \in next(*a'_i)$. Thus, the MR cover approximation for any such place p'_l is found as $\mathcal{C}(p'_l) = \sum \mathcal{C}^*_{t'_k}(p'_l)$ where $\mathcal{C}^*_{t'_k}(p'_l)$ is a cover approximation found for p'_l with a set of concurrent signal instances excluding an instance t'_k immediately preceding t'_j. To reduce the size of MR cover approximations, it is also convenient to choose P'_a so that it includes one input place from each instance in $next(*a'_i)$. The cover approximation for each slice $\mathcal{S}_{On}(+a'_i)$ representing the states from the on-set of signal a_i is therefore calculated as:

$$\mathcal{C}^*_{On}(a'_i) = \mathcal{C}^*_e(+a'_i) + \sum \mathcal{C}^*_{mr}(p'_l),\ p'_l \in P'_a$$

where $\mathcal{C}^*_e(+a'_i)$ may be empty if the entry transition of $\mathcal{S}_{On}(+a'_i)$ is the initial transition of the segment.

Consider approximation of the on-set cover for signal $+a'$ shown in Figure 4(b). The slice representing states from the on-set is found as $\mathcal{S}_{On}(+a') = \langle (p'_1), \{(p'_7, p'_8, p'_9), (p'_6, p'_8, p'_{10}), (p'_5, p'_9, p'_{10})\}\rangle$. To approximate states represented by this slice an approximation set is chosen as $P'_a = \{p'_4, p'_7, p'_{10}\}$. The initial values for MR cover approximations for place p'_4 and p'_7 are found using ξ of their predecessors $+a'$ and $+d'$ respectively. Both places have the same set of concurrent instances of other signals. Their MR cover approximations are found as $\mathcal{C}^*_{mr}(p'_4) = \{1--0--0\} = a\bar{d}\bar{g}$ and $\mathcal{C}^*_{mr}(p'_7) = \{1--1--0\} = ad\bar{g}$. Place p'_{10}, on the other hand, is an input to $-a' \in next(+a')$. Therefore its MR cover approximation is found as $\mathcal{C}(p'_{10}) = \mathcal{C}^*_{f'}(p'_{10}) + \mathcal{C}^*_{e'}(p'_{10}) = \{1--1-01\} \cup \{1--10-1\} = ad\bar{f}g + ad\bar{e}g$. There is only one state in the ER of $+a'$ which is covered by a cover $\mathcal{C}^*(+a') = \{0000000\} = \bar{a}\bar{b}\bar{c}\bar{d}\bar{e}\bar{f}\bar{g}$. The cover approximation representing the on-set of a is found as $\mathcal{C}^*_{On}(a) = \bar{a}\bar{b}\bar{c}\bar{d}\bar{e}\bar{f}\bar{g} + a\bar{d}\bar{g} + ad\bar{g} + ad\bar{f}g + ad\bar{e}g$ (in SOP form).

4.3 Cover refinement

Due to the approximated nature of the covers, an on-set cover found from the STG-unfolding segment may implement an incorrect function. Indeed, if a output signal is implemented using an on-set cover approximation which covers a state belonging to the off-set, then the output will change to "1" where it is suppose to be "0". Thus cover approximations obtained using the algorithm described before need to be checked. To check cover correctness both on- and off-set cover approximations are required.

Suppose that both approximated covers for the on- and off-set of a_i were obtained. Suppose also that their intersection is non-empty. The covers' intersection may only belong to the DC-set. However, to find the DC-set all codes in both on-set and off-set must be known. Therefore, to ensure the covers implement the logic functions correctly we check a stronger condition: *approximated covers for on- and off-set are said to be correct if their intersection is empty.* The approximation produces semi-optimised covers. Exact covers have their intersection empty by construction. Therefore, if the covers' intersection is non-empty, then they need to be refined until their intersection becomes empty, possibly restoring the exact covers. Thus the use of a stronger condition only affects the quality of optimisation rather than correctness of covers. If after complete refinement on- and off-set covers still intersect, then this STG has a CSC problem and cannot be implemented without changes to the specification. Correct refined covers can be optimised using any known minimisation technique using the DC-set, such as Espresso.

A pseudo-code of the algorithm for deriving covers for on- and offsets is shown in Figure 5. The initial on- and off-set cover approximations are found as described in the previous Subsection. If the approximated covers' intersection is not empty, then these covers are refined. Only concurrency relation was used for finding approximated covers. Other relations between transitions concurrent to $*a'_i$ were ignored. The general idea behind refinement is that using these relations some of the information about the cover is restored. Covers are refined until "they are good enough", i.e. covers' intersection becomes empty.

The on- and off-set covers' intersection may become non-empty due to approximation of MR cover for some places in the approximation set. These MR cover approximations may intersect with the ER cover approximations of some instances of the opposite signal transition. In this case only cover approximations for these places (but not all in the approximation set) and the instance of opposite signal transition need to be refined. Further, the set of signals Sig which cause the intersection is also known. These are exactly those signals whose value is undefined in one of the cubes $B \in \mathcal{C}^*$. Thus we need to consider a problem of refining a cover approximation for an element x' of STG-unfolding segment with a set of signals Sig.

To restore some of the relations a *refining set* P'_r is chosen. This set is constructed from non-concurrent places belonging to the slice $\mathcal{S}_{On}(*a'_i)$ such that $\forall p'_k \in P'_r : x' \| p'_k$. Furthermore, the set is chosen so that for at least one signal a_j from Sig for each its instance $t'_k \in \mathcal{S}_{On}(+a'_i)$ one of the successors of t'_k is in P'_r. Thus each refining step will refine at least one signal from Sig. A refined cover $\mathcal{C}^*_{new}(x')$ is obtained from the old approximation as: $\mathcal{C}^*_{new}(x') = \mathcal{C}^*(x') \cdot [\sum \mathcal{C}^{*r}_{mr}(p'_k)], p'_k \in P'_r$. Cover $\mathcal{C}^{*r}_{mr}(p'_k)$ is a restricted MR cover for p'_k

for each implementable signal a_i **do**
    Find set of on-slices $\mathcal{S}^j_{On}$
    for each slice $\mathcal{S}^j_{On}$ **do**
        Find approximation set P'_a
        $\mathcal{C}^*_{On} = \mathcal{C}(t'_e) + [\sum \mathcal{C}^*_{mr}(p'_l) : \ p'_l \in P'_a]$
    end do
    Find set of off-slices $\mathcal{S}^k_{Off}$
    for each slice $\mathcal{S}^k_{Off}$ **do**
        Find approximation set P'_a
        $\mathcal{C}^*_{Off} = \mathcal{C}^k(t'_e) + [\sum \mathcal{C}^*_{mr}(p'_l) : \ p'_l \in P'_a]$
    end do
    /* initial approximations found */
    while $\mathcal{C}^*_{On} \cdot \mathcal{C}^*_{Off} \neq \emptyset$ **then do**
        for each $\mathcal{C}^*_{mr}(p'_l)$ and $\mathcal{C}^k(t'_e)$ such that $\mathcal{C}^*_{mr}(p'_l) \cdot \mathcal{C}^k(t'_e) \neq \emptyset$ **do**
            Find the set of offending signals Sig
            Choose offending signal a_j *from* Sig
            Find refining set P'_r *for* p'_l *w.r.t.* a_j
            $\mathcal{C}^*_{new}(p'_l) = \mathcal{C}^*_{mr}(p'_l) \cdot [\sum \mathcal{C}^{*r}_{mr}(p'_k) : \ p'_k \in P'_r]$
            Find refining set P'_r *for* t'_e *w.r.t.* a_j
            $\mathcal{C}^*_{new}(t'_e) = \mathcal{C}^*(t'_e) \cdot [\sum \mathcal{C}^{*r}_{mr}(p'_k) : \ p'_k \in P'_r]$
        end do
    end do
end do

Figure 5: Algorithm for deriving on- and off-set cover approximations from STG-unfolding segment

where only those literals are set to "–" whose instances $t'_l \| p'_k$ belong to $\mathcal{S}_{On}(*a'_i)$ and are successors of x'.

Informally, at each step the refinement procedure restores the marking component of reachable states represented by the slice. It finds a set of places which can be marked together with each already partially restored marking. The cover function is then changed reflecting the fact that partially restored markings now include found places. Thus in the end, when the procedure terminates, the covers correspond to fully restored markings and cover only states with these marking components.

Since each step refines the value of at least one variable and the set of signals is finite, the refinement procedure will terminate in finite number of steps producing an exact cover for the states represented the slice $\mathcal{S}_{On}(*a'_i)$.

Consider a fragment of STG-unfolding segment shown in Figure 4(c). Suppose that on-set cover approximation $\mathcal{C}^*_{On}$, found with approximation set $P'_a = \{p'_1, p'_3, p'_5, p'_8\}$, intersects with $\mathcal{C}^*_{Off}$ for some signal. Suppose also that a cube $B = d\bar{e}$ which is an MR cover approximation of place p'_5 causes this non-empty intersection. The set of offending signals is found as $Sig = \{a, b, c\}$. Let a be the signal chosen for refinement. Its only instance which should be used in refinement is $-a'$. A refinement set is chosen as $P'_r = \{p'_2, p'_4, p'_7, p'_9\}$.

Benchmark	Sigs	PUNT ACG					Other tools		
		UnfTim	SynTim	EspTim	TotTim	LitCnt	Petrify	SIS	LitCnt
imec-master-read.csc	18	0.39	73.56	3.05	77.00	83	125.66	630.52	69
nowick.asn	7	0.02	0.26	0.69	0.97	17	1.44	0.51	20/17
nowick	6	0.02	0.17	0.38	0.57	15	1.10	0.23	14
par_4.csc	14	0.03	1.12	2.48	3.63	36	12.31	168.55	36
sis-master-read.csc	14	0.16	4.53	1.09	5.78	48	27.09	130.66	48
tsbmSIBRK	25	0.44	37.64	4.62	42.70	72	299.90	141.51	72
pn_stg_example	6	0.01	0.19	1.57	1.77	19	4.20	6.84	19
forever_ordered	8	0.03	0.31	1.12	1.46	20	5.24	8.81	16
alloc-outbound	9	0.05	0.32	0.48	0.85	16	1.75	1.53	16
mp-forward-pkt	20	0.02	0.34	0.47	0.83	17	1.50	0.22	17
nak-pa	10	0.02	0.37	0.57	0.96	20	2.28	0.29	20
pe-send-ifc	17	0.12	1.91	0.50	2.53	68	19.50	1.16	75/72
ram-read-sbuf	11	0.02	0.48	0.58	1.08	25	3.28	0.26	22
rcv-setup	5	0.02	0.06	0.17	0.25	8	0.72	0.14	8
sbuf-ram-write	12	0.04	0.80	0.64	1.48	23	4.04	0.38	23
sbuf-read-ctl.old	8	0.03	0.36	0.47	0.86	15	1.29	0.19	15
sbuf-read-ctl	8	0.02	0.22	0.47	0.71	15	0.99	0.16	15
sbuf-send-ctl	8	0.02	0.37	0.49	0.88	19	1.95	0.21	19
sbuf-send-pkt2	9	0.02	0.49	0.48	0.99	19	2.16	0.23	19
sbuf-send-pkt2.yun	9	0.04	0.58	0.45	1.07	31	3.43	0.26	31
sendr-done	4	0.02	0.02	0.19	0.23	6	0.33	0.14	6
Total	228	1.72	124.10	20.96	146.78	592	520.16	1092.77	580/574

Table 1: Experimental results

Consider calculation of the restricted MR cover approximation for p'_2. The only instances which can be used in approximation is $+e'$ as other concurrent instances, $+a'$ and $+d'$, precede p'_5. Thus $\mathcal{C}^{*r}_{mr}(p'_2) = \{1001-\}$ (the order of signals is $abcde$). Similar, MR cover approximations are found for other places in P'_r. The refined cover approximation is thus found as: $\mathcal{C}^*_{new}(p'_5) = \{---10\} \cap [\{1001-\} \cup \{1101-\} \cup \{1111-\} \cup \{0111-\}] = a\bar{c}d\bar{e} + bcd\bar{e}$.

The resulting cover is an exact cover of MR for place p'_5. Note that if simply MR cover approximation $\mathcal{C}^*_{mr}(p'_2) = \bar{b}\bar{c}$ were chosen for p'_2 (or any other place from P'_r), then refinement would not refine a.

5 Experimental results

The method suggested in this paper was implemented on the basis of the unfolding tool "PUNT". Experiments are divided into two major series.

The goal of the first series was to demonstrate the quality of the proposed method. Results of the synthesis procedure, tested on a set of benchmarks, are shown in Table 1. The table presents time breakdown (in seconds) for synthesis a speed-independent circuit from its STG specification in the atomic complex gate per signal architecture ("PUNT ACG"). Columns "UnfTim", "SynTim" and "EspTim" show times taken to construct the STG-unfolding segment, derive on- and off-set covers and apply Espresso to optimise the covers, respectively. Column "TotTim" shows the total time taken to synthesise a particular circuit. For comparison, same set of benchmarks was synthesised using two known tools Petrify and SIS. Their timings are grouped in the column "Other tools". Literal count (columns "LitCnt") was used as a measure of the quality of the new synthesis method. The literal count shows the total number of literals in the obtained covers of final implementations. The number of signals (column "Sigs"), influencing the complexity of the specification and its behavioural representation, is also given for each specification.

As it can be observed, the synthesis technique based on the STG-unfolding segment produces implementations comparable with those produced by other tools. The timing results show that our technique compares favourably to Petrify. It is also comparable with SIS on the benchmarks with low count of signals and it becomes increasingly better with the

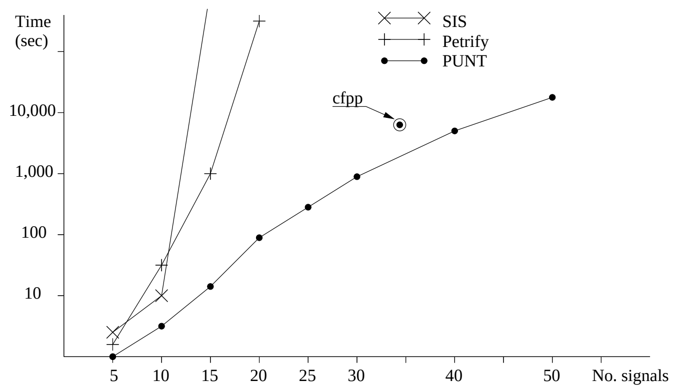


Figure 6: Experimental results for Muller pipeline.

growth of the signal count. These results show that for small sized benchmarks, the overheads of constructing the STG-unfolding segment and traversing it may outweigh the time spent on constructing a small reachability graph with an efficient implementation. Using a stronger correctness condition for approximated covers may produce a slightly worse implementation due to the fact that the DC-set is partitioned.

The second series of experiments shows the feasibility of the new method on a set of scalable examples. The Muller pipeline was chosen to illustrate the ability of the new method. Experimental results are shown in Figure 6. As can be observed, existing tools soon choke on the size of the specification either running out of memory or taking prohibitively long time. Both SIS and Petrify exhibit doubly exponential growth of time taken. The first dependency is due to the state space explosion, the second is due to the exponential complexity of the exact synthesis process used in both tools. In addition, we synthesised a Counterflow pipeline specification [13] which has 34 signals. From the existing tools, only Petrify was able to synthesise it taking more than 24 hours. At the same time PUNT was able to synthesise it in under 2 hours thus giving an order of magnitude gain in speed. This is shown on the graph as a circled dot.

6 Conclusions

In this paper we presented a new method for synthesis of speed independent circuits. Our approach is based on the STG-unfolding segment. It uses the segment as a model from which an implementation is obtained. As the size of the STG-unfolding segment is often smaller than the size of the SG, it is possible to synthesise specifications of larger sizes. In addition, due to the smaller size of the semantic model, the implementation can be achieved faster on a number of moderate sized examples. We demonstrated applicability of our method on an existing set of benchmarks.

Future developement of this method can be directed into exploring heuristics for the refinement procedure. In addition, this method can be adapted to the other implementation architectures. In this case, the approximation will be targeted at obtainning the excitation functions for memory elements. Furthermore, the method can be enhanced by accomodating checks for weaker correctness conditions for approximated covers. It is possible to find a set of slices of the STG-unfolding segment where a particular cover becomes TRUE. Thus if an approximated on-set cover becomes TRUE within slices of the off-set cover approximation (and vice versa), then further refinement is required. Thus, even if the covers intersection is non-empty, but they do not become TRUE within the slices of the opposite cover, then their intersection belongs to the DC-set. This allows to check the cover correctness condition introduced in Section 2 and to obtain covers with better literal counts.

References

[1] T.A. Chu. *Synthesis of Self-Timed VLSI Circuits from Graph-theoretic Specifications.* PhD thesis, MIT, 1987.

[2] Jordi Cortadella, Michael Kishinevsky, Alex Kondratyev, Luciano Lavagno, and Alex Yakovlev. Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. In *Proc. of the 11th Conf. Design of Integrated Circuits and Systems*, Barcelona, Spain, November 1996. to appear.

[3] M. Kishinevsky, A. Kondratyev, A. Taubin, and V. Varshavsky. *Concurrent Hardware: The Theory and Practice of Self-Timed Design.* John Wiley and Sons, London, 1993.

[4] K.L. McMillan. *Symbolic Model Checking.* Kluwer Academic Publishers, Boston, 1993.

[5] T.H.-Y. Meng, R.W. Brodersen, and D.G. Messershmitt. Automatic synthesis of asynchronous circuits from high-level specifications. In *IEEE Transactions on Computer-Aided Design*, volume 8(11), pages 1185–1205, 1989.

[6] T. Miamoto and S. Kumagai. An efficient algorithm for deriving logic functions of asynchronous circuits. In *Proc. of the Second International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC'96)*, pages 30–35, Aizu-Wakamatsu, Fukushima, Japan, March 1996.

[7] Enric Pastor and Jordi Cortadella. An efficient unique state coding algorithm for signal transition graphs. In *Proc. of the IEEE International Conference on Computer Design*, pages 174–177, USA, October 1993.

[8] Enric Pastor, Jordi Cortadella, Alex Kondratyev, and Oriol Roig. Structural methods for the synthesis of speed-independent circuits. In *Proc. European Design and Test Conference (EDAC-ETC-EuroASIC)*, pages 340–347, Paris(France), March 1996.

[9] Wolfgang Reisig. *Petri Nets, An Introduction.* Springer-Verlag, 1985.

[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.

[11] A. Semenov and A. Yakovlev. Event-based framework for verification of high-level models of asynchronous circuits. Technical Report 487, University of Newcastle upon Tyne, 1994.

[12] E.M Sentovich and et. al. SIS: A system for sequential circuit synthesis. Memorandum No. UCB/ERL M92/41, University of California, Berkeley, 1992.

[13] A. Yakovlev. Designing control logic for counterflow pipeline processor using petri nets. Technical Report 522, University of Newcastle upon Tyne, 1995.

[14] C. Ykman-Couvreur, Bill Lin, Gert Goossens, and Hugo De Man. Synthesis and optimization of asynchronous controllers based on extended lock graph theory. In *Proc. European Design and Test Conference (EDAC-ETC-EuroASIC)*, pages 512–517, February 1993.

[15] Ch. Ykman-Couvreur, B. Lin, and H. DeMan. ASSASSIN: A synthesis system for asynchronous control circuits. Reference manual, IMEC, 1995.