

Verification driven synthesis of asynchronous circuits from STG specification

Ilya V. Klotchkov Alexander B. Smirnov Nikolai A. Starodoubtsev

1 Introduction

The main goal of this work is to develop an efficient algorithm for synthesis of asynchronous control circuits from Signal Transition Graph (STG) specifications introduced in [1]. There are state space (State Graph) exploration based [2, 3, 4] as well as structural [5, 6] methods of asynchronous control circuits synthesis known up to date. They suffer either from the exponential processing time or non-optimal solution. In this paper we consider a method which allows taking advantage of both: optimal solution, obtained with SG based methods and fast processing time of structural approaches. Advantage of this approach is most obvious for the STGs that allow the negative or simple gate implementation.

2 Terms and definitions

2.1 Basic definitions

A marked Petri net (PN) $\mathcal{N} = \{\mathcal{P}, \mathcal{T}, \mathcal{F}, m_0\}$ is a directed bipartitioned graph, where \mathcal{P} and \mathcal{T} are partitions (elements (nodes) in \mathcal{P} and \mathcal{T} are called places and transitions correspondingly), $\mathcal{F} = (\mathcal{P} \times \mathcal{T}) \cup (\mathcal{T} \times \mathcal{P})$ is a set of arcs (called flow relation) and $m_0 \in \mathcal{P}$ is initial marking. Denote the sets of predecessors and successors of a node n as $\bullet n$ and $n \bullet$ respectively. Similarly for an arc e $\bullet e$ and $e \bullet$ are its source and destination. PN operation rules are as follows: a transition can **fire** or is **enabled** iff all its preceding places are marked. A transition firing removes one marker from every preceding place and puts one marker on every successive place. A marking m is **reachable** from the initial marking m_0 iff there exists a feasible sequence of transition firings which leads to m . A PN is called N -bounded iff at any reachable marking a number of markers in any place does not exceed N . Let $A = \{a_i\}_{i=1}^N$ be a set of N signals. STG is an interpreted PN, where interpretation binds transitions $t \in \mathcal{T}$ with labels from the set of signal transitions $A \times \{+, -\}$. We use notations $a+$ and $a-$ for PN transitions instantiating rising and falling transitions of some signal a . An STG example is given in the Figure 1.

Consider the PN, underlying an STG. An STG transition t_j is said to be **locally concurrent** (or concurrent in the given marking) to a place p_i if there exists a reachable marking m where t_j is enabled, p_i being marked and $p_i \notin \bullet t_j$. Denote the set of STG signals, which transitions are concurrent to a place p in the given marking m as $\mathbb{C}_m(p)$. An STG place p_i is **concurrent** to a place p_j if exists m such that p_i and p_j are both marked. The set of places concurrent to p denote as $\mathbb{C}(p)$. STG specification is **implementable** if it is **live** (no transition is permanently disabled), **bounded**, **consistent** (sequence transitions of each signal alternate in any firing sequence), **output persistent** (no transition firing disable other transitions of non-input signals) and satisfy the Complete State Coding (**CSC**) property [2]. The latest means that for any two reachable markings with the same state the set of enabled transitions of non-input signals must be the same.

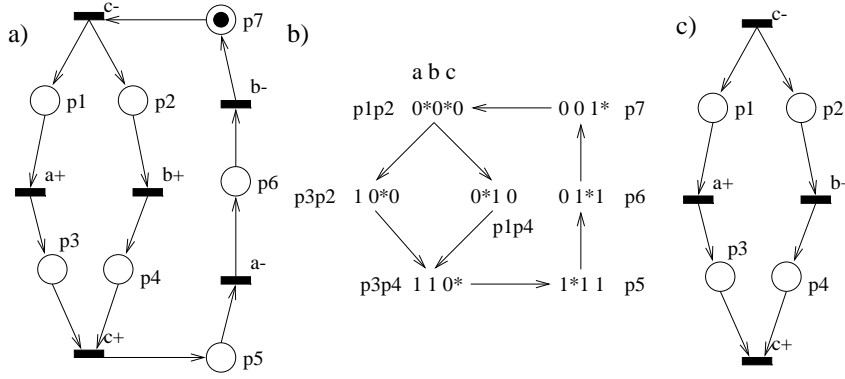


Figure 1: STG example (a), its state/reachability graph (b), interval for c^- (c).

2.2 Partial states

Consider an STG place p . The fact that it is marked provides the STG total state (the state of all STG signals) iff no STG signal transitions are concurrent to that place. As it could be seen from the reachability graph of the net from the Figure 1, the place p_7 represents the total state of this STG (001), while p_1 does not. Marked p_1 gives a set of possible STG total states $\{000, 010\}$. Thus, only STG complete marking completely defines its total state. For a Boolean function let a **literal** be a variable(signal) or its complement, a **cube** be a product of literals. Each state s can be represented as a cube $c(s)$. For instance the state (010) for the STG in the Figure 1 gives cube $\bar{a}b\bar{c}$. Later on we do not distinguish states and their representations by cubes. Define a projection of the total state s on some set of signals T as an operation $\mathbf{Pr}_{\downarrow T}(c(s))$, which removes literals of all the signals $a \notin T$ from the $c(s)$. Following the example above, $\mathbf{Pr}_{\downarrow\{a,b\}}(\bar{a}b\bar{c})$ is $\bar{a}b$. For each place $p \in \mathcal{P}$ denote a partial states' set of STG as follows:

Definition 2.1 Let p be a place in the STG, m be a marking where p is marked and s be a vertex in the SG, obtained by encoding m . Define projection $\mathbf{Pr}_{\downarrow A \setminus \mathcal{C}_m(p)}(c(s))$ to be a partial state in marking m , which corresponds to the place p .

Generally, there may be several different reachable markings where p is marked, so projections $\mathbf{Pr}_{\downarrow A \setminus \mathcal{C}_m(p)}(c(s))$ for any particular place may differ.

Definition 2.2 An STG satisfies Unique Partial State (UPS) property iff : $\forall p \in \mathcal{P}$ and $\forall m_i, m_j$ such that $p \in m_i$ and $p \in m_j$ holds $\mathbf{Pr}_{\downarrow A \setminus \mathcal{C}_{m_i}(p)}(c(s)) = \mathbf{Pr}_{\downarrow A \setminus \mathcal{C}_{m_j}(p)}(c(s))$, i.e. partial state is uniquely defined. In that case we call the projection a **partial state** and denote $\mathbb{P}(p) = \mathbf{Pr}_{\downarrow A \setminus \mathcal{C}_{m_j}(p)}(c(s))$.

The property 2.2 can be checked on the state graph of the STG by explicit projection of the states on the places. However, the state graph is not necessary for checking UPS property. Concurrency information for each place would be sufficient. It is obtained by the reduced STG¹ symbolic traversal [4]. Below we call an STG **canonical** iff it is implementable, safe (1-bounded) and satisfies UPS property. For each signal a_i of the canonical STG we define an equivalence relation, assuming that the projection operator can be applied to any cube (not only to the total states):

Definition 2.3 Let a_i be an STG signal. Equivalence relation $\rho_{a_i} \subseteq \mathcal{P} \times \mathcal{P}$ binds two places $(p_1, p_2) \in \rho_{a_i}$ iff in their partial states the signal a_i is in the same state: $\mathbf{Pr}_{\downarrow a_i}(\mathbb{P}(p_1)) = \mathbf{Pr}_{\downarrow a_i}(\mathbb{P}(p_2)) \neq \emptyset$. For each place p a conjugacy class $\rho_{a_i}(p)$ induced by ρ_{a_i} is a places' set equivalent to p , i.e. $\rho_{a_i}(p) = \{p' \in \mathcal{P} | (p, p') \in \rho_{a_i}\}$.

¹A PN obtained from the original STG by collapsing all STG linear branches to reduce the reachable states' space.

The definition 2.3 induces a set of classes on the original STG. For each a_i two classes represent sets of places where the signal is 1 and 0 respectively. All the places where signal a_i is not defined in the partial state form single element sets relative to equivalence relation ρ_{a_i} .

Definition 2.4 An **interval** \mathcal{I}_{l_i} of a literal l_i is a conjugacy class of some place where the signal a_i is met in the appropriate polarity: $\mathcal{I}_{l_i} = \rho_{a_i}(p)$ such that $\mathbf{Pr}_{\downarrow\{a_i\}}(\mathbb{P}(p)) = \delta(l_i)$, where $\delta(a_i) = 1$ and $\delta(\bar{a}_i) = 0$.

The Figure 1 (c) shows an interval for literal \bar{c} for the STG from (a). Finally, an example of non-canonical STG is given in the Figure 2 a). Here the partial state for the place $p7$ is not uniquely defined. If $p7$ is marked after $c+$ firing, the $p7$ is concurrent in that marking to $a-$ and $b+$. If $p7$ is marked after $b+$ firing it is not concurrent to any transition. For the comparison, Figure 2 b) shows the same behavior specified by a canonical STG.

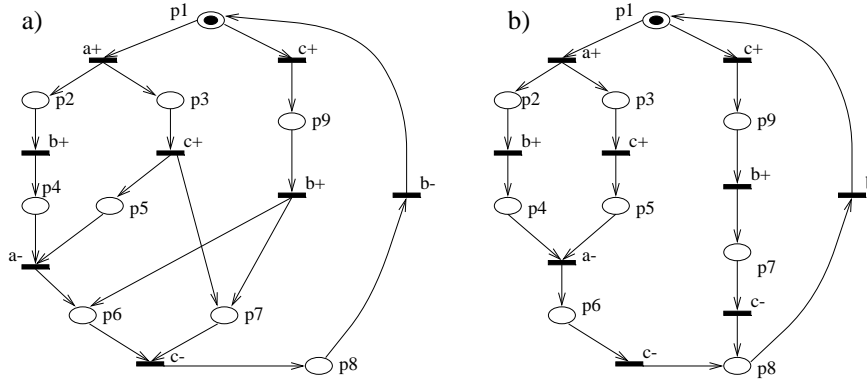


Figure 2: Non-canonical STG (a) and equivalent canonical STG (b).

2.3 Compatible paths and intervals

A **multipath** $\mathfrak{T} \subseteq \mathcal{F}$ in a directed graph is a set of arcs $\{e_i\}$. A multipath \mathfrak{T} generates a subgraph $G_{\mathfrak{T}}$ as the following pair: $G_{\mathfrak{T}} = (\{n_i | \exists e_k \in \mathfrak{T} : n_i = \bullet e_k \text{ or } n_i = e_k \bullet\}, \mathfrak{T})$. Similarly, on a given set of nodes V a multipath $\mathfrak{T}(V)$ may be generated: $\mathfrak{T}(V) = \{e_i | \exists n_i, n_j \in V : n_i = \bullet e_i, n_j = e_i \bullet\}$. For each multipath \mathfrak{T} entry and exit sets could be defined:

Definition 2.5 A **multipath entry set** $\partial_{in}\mathfrak{T} = \{n_k \in \mathcal{P} \cup \mathcal{T} \mid \nexists e_j \in \mathfrak{T} : e_j \bullet = n_k \text{ and } \exists e_i \in \mathfrak{T} : \bullet e_i = n_k\}$ A **multipath exit set** $\partial_{ex}\mathfrak{T} = \{n_k \in \mathcal{P} \cup \mathcal{T} \mid \nexists e_j \in \mathfrak{T} : \bullet e_j = n_k \text{ and } \exists e_i \in \mathfrak{T} : e_i \bullet = n_k\}$

A multipath \mathfrak{T} is called a **path** iff $|\partial_{in}\mathfrak{T}| = |\partial_{ex}\mathfrak{T}| = 1$ and is denoted \mathcal{C} . Entry and exit sets of a path are called start and end respectively. A path $\mathcal{C}(n_1, n_2)$ **connects** node n_1 with node n_2 iff $n_1 = \partial_{in}\mathcal{C}$ and $n_2 = \partial_{ex}\mathcal{C}$. An interval from the definition 2.4 can not generate a non-empty multipath since no nodes in the interval are adjacent. However, a set of places which forms an interval could be extended by adding a set of appropriate transitions to generate a multipath:

Definition 2.6 Denote $\mathfrak{T}'(\mathcal{I}_{a_i})$ a multipath generated by an interval \mathcal{I}_{a_i} : $\mathfrak{T}'(\mathcal{I}_{a_i}) = \mathfrak{T}(\mathcal{I}_{a_i}) \cup \{t_k \in \mathcal{T} \mid \exists p_l, p_m \in \mathcal{I}_{a_i} : t_k \in p_l \bullet \text{ and } t_k \in \bullet p_m\} \cup \{e \in \mathcal{F} \mid e \bullet = a_i - \text{ or } \bullet e = a_i +\}$

Definition 2.7 Two multipaths are **compatible** $\mathfrak{T}_1 \rightleftharpoons \mathfrak{T}_2$ iff $\partial_{in}\mathfrak{T}_1 \equiv \partial_{in}\mathfrak{T}_2$ and $\partial_{ex}\mathfrak{T}_1 \equiv \partial_{ex}\mathfrak{T}_2$

3 Interval hull

Given a signal a_i , the STG partitioning could be defined as follows:

Definition 3.1 Signal partition (or *S-partition*) of the STG relative to a signal a_i is a set of five subgraphs $SP_{a_i} = \{QR^1(a_i), QR^0(a_i), ER^1(a_i), ER^0(a_i), \mathcal{R}(a_i)\}$ where subgraphs are defined as follows :

- $QR^1(a_i)$ and $QR^0(a_i)$ are quiescent regions of signal a_i , where it is stable at 1 or 0 respectively : $QR^1(a_i) = \{p_j \in \mathcal{I}_{a_i} \mid a_i - \notin p_j \bullet\}$, $QR^0(a_i) = \{p_j \in \mathcal{I}_{\bar{a}_i} \mid a_i + \notin p_j \bullet\}$
- $ER^1(a_i)$ and $ER^0(a_i)$ are excitation regions of signal a_i , where a_i is excited to change its value : $ER^1(a_i) = \{p_j \in \mathcal{I}_{\bar{a}_i} \mid a_i + \in p_j \bullet\}$, $ER^0(a_i) = \{p_j \in \mathcal{I}_{a_i} \mid a_i - \in p_j \bullet\}$
- $\mathcal{R}(a_i)$ is the remainder : $\mathcal{R}(a_i) = \mathcal{P} \setminus (QR^1(a_i) \cup QR^0(a_i) \cup ER^1(a_i) \cup ER^0(a_i))$ – all the places in STG which are not in any other set of the partition.

The definition 3.1 gives STG counterpart of the well known SG quiescent and excitation regions. Based on the signal partition we define a next-state function for each signal a_i :

Definition 3.2 Let p_j be a place in the STG and a_i be a signal. **Next-state function** of the a_i in p_j is defined as follows :

$$a_i'(p_j) = \begin{cases} 1 & \text{if } p_j \in ER^1(a_i) \cup QR^1(a_i) \\ 0 & \text{if } p_j \in ER^0(a_i) \cup QR^0(a_i) \\ - & \text{(undefined) if } p_j \in \mathcal{R}(a_i) \end{cases}$$

The remaining set $\mathcal{R}(a_i)$ is a set of places where the signal a_i is not defined. That set should be further partitioned in order to allow for the more accurate synthesis, for what we introduce the notion of interval hull. Define a **subcut** as a set of concurrent places following terminology from [5]. A maximal subcut is called a **cut** in [5]. An **excitation subcut** of a given transition t_i is a minimal set of places, which must be marked in order to fire t_i . Consider first an excitation subcut C of t_i . By the PN operation rules, there is a set of transitions $\{t_j\}$ which firings put markers on the places in C . Firing a set of transitions $\{t_j\}$ puts markers on a subcut $\bigcup(t_j \bullet) = C'$ which could be larger than C . We call C' an **adjoint subcut** of C . Generally an adjoint subcut is not unique for C . However, it can be proved that in a canonical STG adjoint excitation subcut of a persistent transition is unique, as it follows from the UPS property. An adjoint subcut is illustrated in the figure 3. For the transition $d+$ excitation subcut contains only the place $p1$ while its adjoint excitation subcut is $\{p1, p2\}$. For each signal transition we define a pair function :

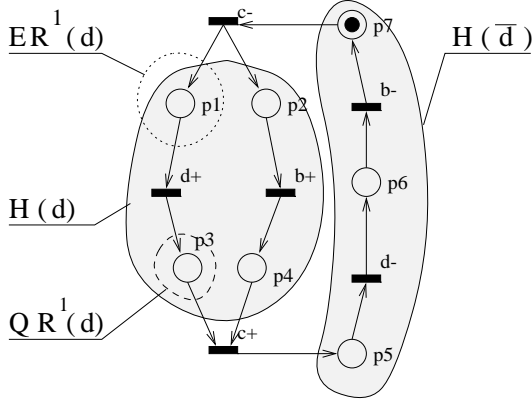
Definition 3.3 Let t_i be a transition of signal a_j and l_j be a literal of that signal. Then a **brink pair function** $\mathcal{E}(t_i)$ gives a set of opposite direction transitions of the signal a_i , which are connected within the interval \mathcal{I}_{l_j} with the transition t_i : $\mathcal{E}(t_i) = \{t_j \in \mathcal{T} \mid \exists \mathfrak{C} \in \mathfrak{T}'(\mathcal{I}_{l_j}) : \text{path } \mathfrak{C} \text{ connects } t_i \text{ with } t_j\}$

Definition 3.4 Let us assume that $\{t_j\}$ is a set of rising(falling) transitions of certain signal a_i , l_i is its positive(negative) literal and $\{C_j\}$ is a set of their excitation subcuts. Then **hull entry set** $\partial H(l_i)$ of a literal l_i is $\partial H(l_i) = \bigcup (C_j' \setminus \mathbb{C}(\bullet \mathcal{E}(t_j)))$.

Definition 3.5 A **hull** $H(l_i)$ of a literal l_i is a set of places such that :

$$H(l_i) = \partial H(l_i) \cup \left\{ p \in \mathcal{P} \left| \begin{array}{l} \exists p' \in \partial H(l_i) \text{ and } \exists \mathfrak{C}_1(p', p) \\ \exists \mathfrak{C}_2(p, p'') : p'' \in \mathcal{E}(\bullet p') \text{ and} \\ \mathfrak{T}'(\mathfrak{C}_1 \cup \mathfrak{C}_2) \cap (\mathbb{C}(\mathcal{E}(\bullet p')) \cup ER^{\delta(\bar{l}_i)}(l_i)) \equiv \emptyset \end{array} \right. \right\}$$

In other words, the hull of l_i is a closure of the hull entry set by the relation of non-concurrency to opposite excitation set $ER^{\delta(\bar{l}_i)}(l_i)$. For the sake of convenience we consider positive and negative hull of interval to be the hull of the corresponding literals.



An interval hull specifies a subgraph in the STG, to which a corresponding excitation and quiescent regions may be extended in order to obtain optimal solution, as it is shown later. The Figure 3 illustrates the above definitions. $ER^1(d)$ is the place p_1 . However, subcut $\{p_1\}$ has an adjoint subcut $\{p_1, p_2\}$ which is hull entry set $\partial H(d)$. The hull $H(d)$ contains also places p_3 and p_4 , but not the place p_5 , because the latter belongs to $ER^0(d)$ that is not allowed by the third condition in the definition 3.5.

Figure 3: Regions and hull for signal a .

4 Implementation strategy

The synthesis approach described in this paper is based on iterative probing of prospective implementations of every STG signal with subsequent verification. The search is performed in the constrained (by the hull) space of Boolean functions. The selection of solution is based on the verification criteria, which is applied to all prospective solutions in order to identify correct ones. As it is shown later, the verification is based on the path compatibility property and the mapping of Boolean functions into generalised intervals that is discussed in the next section.

4.1 Boolean algebra representation in the intervals set

In the section 2.2 intervals to literals correspondence is shown. Consider the mapping of a limited class of Boolean functions into the set of intervals. Boolean AND of two literals a and b may be interpreted as a set of places in the STG where both signals yield appropriate value, thus resulting in a set-theory intersection of the corresponding intervals: $a \wedge b \Leftrightarrow \mathcal{I}_a \cap \mathcal{I}_b$. In turn, the Boolean OR maps to the set-theory union of intervals: $a \vee b \Leftrightarrow \mathcal{I}_a \cup \mathcal{I}_b$. A set of all STG intervals may be closed by adding all possible intersections and unions of intervals. In the resulting set \mathfrak{I}' operations \cup and \cap form algebraic structure that is homomorphous to binary algebra on the set of signals. The remaining unary Boolean operation ($\bar{}$) does not allow direct mapping, as $\mathcal{I}_a \cup \mathcal{I}_{\bar{a}} \neq \mathcal{P}$. On the other hand, since both literals of each signal have their counterparts in the set of intervals, negative variables of Boolean functions can still be mapped to intervals. For instance, a function $a + b\bar{c}$ maps to $\mathcal{I}_a \cup (\mathcal{I}_b \cap \mathcal{I}_{\bar{c}})$. Thus, we formally define representation of Boolean **not** in the closed set of intervals \mathfrak{I}' as operation of taking interval for opposite polarity literal. That definition holds only for literals, since the negation operation in \mathfrak{I}' can not be closed and thus is not a Boolean negation. Thus, Boolean function on STG signals reduced to the sum-of-products form may be mapped into a set of places $\{p_i\} \in \mathcal{P}$. According to the Boolean inversion formal definition, De Morgan law does not hold for \mathfrak{I}' .

4.2 Verification criteria

In the previous sections it is shown that Boolean operations map onto the set of intervals. Consider logic functions f in the sum of product form with all inversions being only applied to single variables. Any logic function may be represented in such a form, and then mapped

into the \mathfrak{I}' . Define an **on-set** $S_f(1)$ (**off-set** $S_f(0)$) as a set of places where the function evaluates to 1 (0). On the remaining set $\mathcal{P} \setminus (S_f(1) \cup S_f(0))$ the function f is not defined. **Implementation** of a literal $l_i = a_i$ ($l_i = \bar{a}_i$) is a partially defined logic function that evaluates to 1 on the set of places where the next state function a'_i also evaluates to 1 (0). The following statement limits possible implementations for a given literal l_i .

Theorem 4.1 *A logic function f implements a literal l_i of a signal a_i iff*

1. $\mathfrak{I}'(S_f(1))$ is compatible with $\mathfrak{I}'(H(l_i))$
2. $S_f(1) \subseteq H(l_i)$
3. $\forall p \in H(l_i)$, $e \in \mathfrak{I}'(S_f(1))$ such that $|\bullet p| > 1$ or $|p \bullet| > 1$ and $e \bullet = p$ or $\bullet e = p$ for any arc $f : f \bullet = p$ or $\bullet f = p$ respectively, must hold $f \in \mathfrak{I}'(S_f(1))$.

Sketch of the proof. In the canonical STG excitation and quiescent regions do not intersect. Next, as stated in the section 2.3, two paths are compatible if their entry and exit sets are the same. Hence, $\mathfrak{I}'(S_f(1))$ forms a set of connected paths in the $H(l_i)$ such that each node in the entry set is connected with some node from the exit set. For any particular feasible sequence in $H(l_i)$, the function f must remain at a predefined state if it implements literal l_i . In these sequences two classes are clearly distinguished: concurrent and conflicting places. We call two places **conflicting** iff they are neither concurrent nor sequential. For any two concurrent places $S_f(1)$ must cover at least one of them as they lay on two paths, executed concurrently. All conflicting places must be covered by $S_f(1)$ since their activation is mutually exclusive. Therefore, if $\mathfrak{I}'(S_f(1))$ includes some arcs adjacent to multi fan-in or fan-out places it must also include all such arcs. That selection is controlled by the condition 3, which states that for all multi fan-in and fan-out places all possible paths starting or ending in those places must be covered by $S_f(1)$. On the other hand, the condition 2 guarantees the correct polarity of the function, as its on-set can not cover $ER^0(a_i)$. However, $H(l_i)$ superset $QR^1(a_i)$, as it includes places, which are sequential with adjoint excitation subcut of a_i+ . These places may be covered by an implementation function, as they belong to a path concurrent to $\mathfrak{I}'(\mathcal{I}_{a_i})$ and therefore if $S_f(1)$ covers them f implements a_i correctly ■

The theorem below states that the search space of the prospective implementation of the literal l_i can not be extended any further:

Theorem 4.2 *An interval hull $H(l_i)$ is maximal ambient set which must contain on-sets of all possible implementations of the literal l_i . Dual statement holds for $H(\bar{l}_i)$.*

Sketch of the proof. For a canonical STG, an interval hull $H(l_i)$ represents a maximal set of places, which are sequential with an adjoint excitation subcut of the transition of a_i , are neither concurrent to $\{ER^0(a_i)\}$ nor overlap with $\{ER^0(a_i) \cup QR^0(a_i)\}$. None of these requirements can be relaxed. If the hull is extended by sequential (with hull's nodes) places then it overlaps with negative excitation region and the implementation function will remain stable when it must be exited. Neither can it be extended by conflicting or concurrent to hull's nodes, because then implementation function may become undefined at some point ■

Summarizing the two theorems above, f implements a literal l_i if:

1. on-set multipath $\mathfrak{I}'(S_f(1))$ is compatible with $\mathfrak{I}'(H(l_i))$
2. on-set multipath belongs to a corresponding hull: $\mathfrak{I}'(S_f(1)) \subseteq \mathfrak{I}'(H(l_i))$
3. for multi fan-in and fan-out places either all or no adjacent arcs belong to the on-set multipath

4.3 Synthesis methodology

The criteria from the section above may be utilized for logic synthesis. Following the terminology from [5] we consider the complex gate per signal implementation. In that case the algebra \mathcal{J}' from section 4.1 is not sufficient: an implementation of positive literal for some signal a_i must also implement the negative literal of the signal. That is not guaranteed by the \mathcal{J}' since the De Morgan law does not hold for \mathcal{J}' . Therefore we require the representation of implementation function for a_i to satisfy the De Morgan law. I.e. if the function implements a literal, then its De Morgan transformation must implement the complement.

Among the implementations which have passed verification the lowest weight solution is selected as the best solution. In the proposed methodology the weighting is used to account the target technology. For instance, generic two-level gates library would have the following weight function: $W = \sum_i a_i * l_i$ where a_i are weights of each input signal to the gate and l_i are level penalties, showing how the weight of a single signal depends on the logic level. Assuming that all signals have the same weight of 4 and the level penalty is 3, the 2AND-2OR gate weight is $(4 + 4) * 3 + 4 = 28$.

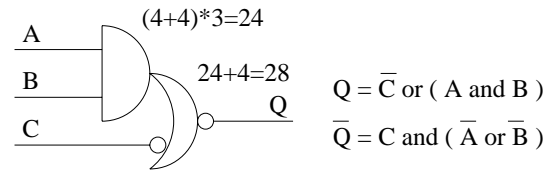


Figure 4: Weight function example.

Now we have to estimate the synthesis algorithm complexity. First, we need to identify how many different implementation forms are available in a given basis. Since each STG signal is represented by two intervals, functions that differ only in inversion on output or some inputs are considered equal. Designate n as a number of different function's upon m variables in the given technology basis. A brute force search algorithm (without heuristic) has the complexity of $O(nN^m)$, where N is a number of literals of the STG. Next, verification of each prospective implementation (by the criteria above) grows roughly linearly with size of STG when implemented as a width search algorithm [8]. Finally, mapping of a function into its on-set (off-set) in the STG has complexity $O(N)$. Therefore, the total complexity is $O(nN^{m+1})$. That is the worst case estimation. Usually each signal has at least two STG transitions (rising and falling). These depend immediately on some signals' transitions, which corresponding literals must always be in the implementation. Thus, at least one term (called driving) is known and does not need to be searched. In average case this gives complexity reduction down to the $O(nN^{m-1})$. Applying a good heuristic with branch cutoff reduces the average case computing time significantly. The solution space exploration may be significantly optimised in the case the gate library is limited (i.e. generic ASIC or standard-cell technology).

4.4 Two-level gates heuristic

If the library contains only two-level gates and the widest gate has four forms with 6 inputs at most, a rough estimation of the worst-case complexity gives $O(4N^5)$. However, for the two level gates a branch cut-off heuristic exists. Starting from the driving terms a tree search algorithm may be used to find an implementation. Namely, assuming driving terms set to be the tree root, we may iteratively append terms to the root obtaining new nodes. The search continues until a correct implementation is obtained. That implementation is recorded and the branch cutoff is initiated based on the solution weight found. Such an algorithm does not limit the search space but provides a way to quickly find the best solution in the given technology.

4.5 Speed independence and place hook

This section describes the class of synthesized circuits. The verification criteria above states that implementation function of some signal is guaranteed to have appropriate value in each place of the corresponding regions. Transitions are assumed to fire instantly, what is not true for real-life circuits. Therefore, it is necessary to guarantee the speed independence of the synthesized circuits. To guarantee speed independence we require 'place hook' between terms of the implementation function for each signal. That is, we require every interval that participates in the cover to have a common place (and therefore a common state) with at least one other interval of the cover. Assuming unbounded gate delays, the intervals' common place guarantees that successive transition of the common place will never fire before preceding transition. Thus we make sure that the transitions' firings order in the circuit does not depend on the time the transitions may take to fire. Consider an example from the Figure 5. The function $a \vee b$ which must evaluate to 1 on $\mathcal{I}_a \cup \mathcal{I}_b$. It is held in 1 in places $p1, p2, p3$ by the term a and in places $p3, p4, p5$ by the term b , their intersection being $p3$. This requirement limits the search space, increasing the overall processing speed. The place hook guarantees the speed independence of the synthesized circuit under unbounded gates delay assumption and allows extraction of the fork isochronity conditions for the synthesised circuits. For instance, assuming that function $a \vee b$ from the Figure 5 implements signal x , we can say that event $a-$ must not arrive to target gate after the event $b+$. This gives a fork isochronity condition for the circuit layout : $\Delta_{t_{b+}, t_{a-}} > \tau_{\{b_{out} \rightarrow x_{in}\}} - \tau_{\{a_{out} \rightarrow x_{in}\}}$, where $\Delta_{t_{b+}, t_{a-}}$ is time between the events and $\tau_{\{a_{out} \rightarrow x_{in}\}}$ is wire transport delay.

4.6 Solution optimality

In the theorem 4.2 it was proved, that the hull is a maximal set of places to which implementation functions images must belong to in order to provide a correct implementation. In combination with the verification driven exploration of the solution space this provides the optimal solution **within** our methodology. Because all Boolean function from the subset of all Boolean functions (as defined by the target technology) are probed, the optimality can only be compromised by solution selection criteria. There are three main concerns on the methodology optimality:

1. how adequate is the intervals algebra \mathfrak{I}' representation of the possible states reached by the STG during modeling, i.e. are there SG states that are omitted by the proposed methodology;
2. is the function mapping to \mathfrak{I}' optimal;
3. place hook condition from the section 4.5 limits possible implementations of each signal. That however contributes to circuit robustness and simplifies the synthesised circuit decomposition, as intermediate terms in the complex gates already satisfy local verification criteria.

The first item is easily reduced to the second item, as if the algorithm performs exhaustive exploration of the solution space, the only inoptimality may be introduced by the selection criteria. It is clear, that mapping proposed in the section 4.1 is not the only possible. For example, mapping of the Boolean OR may be redefined to include all the places outside the corresponding intervals' union, where one of the two signals is guaranteed to have appropriate value. This is illustrated in the figure 5, there the shaded area shows the places where the function $a \vee b$ evaluates to 1. Apparently that area exceeds the union of intervals $\mathcal{I}_{a+} \cup \mathcal{I}_{b+}$, which may cause solution inoptimality.

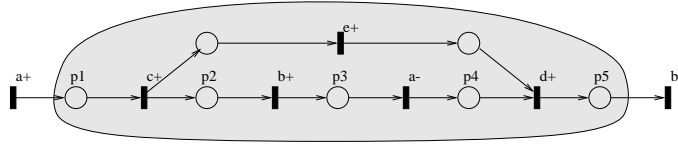


Figure 5: Extended mapping of Boolean OR into \mathcal{J}' .

5 Experimental results

The synthesis algorithm is implemented in a CAD tool ‘TaxoSynthesis’, which is being used in experiments with asynchronous benchmarks [2]. In the Table 1 the processing time and the synthesized circuit complexity is compared to that of the Petrify (asynchronous circuits’ CAD tool). The time measurements were made for Petrify with the “time” command on Linux Slackware 3.1 (kernel 2.0.0). Those for TaxoSynthesis were made with use of “pview” application. Both measurements were made on the same (dual-boot) Pentium-100 32MB RAM PC. The comparison shows that for the small examples the two systems are not much different. The results dispersion comes from two major factors: 1) time measurement inaccuracy under Windows and 2) difference in the implementation strategy between two systems, TaxoSynthesis works solely with logic functions, while Petrify targets simple C-elements. Several examples exhibit TaxoSynthesis inoptimalities, namely inoptimal algebra mapping and the place hook criteria being excessive for the speed-independence. The two downmost rows of the table show the advantage the proposed approach. In that case, two benchmarks were to allow simple gates implementation, which made them considerable large. Since those two graph are inherently concurrent, processing time of the state-based Petrify tool has risen. On the other hand, processing time of the TaxoSynthesis has risen much less according to its polynomial law. To make the comparison fair, it must be noticed that some examples (for instance, vme* serie of benchmarks) do not allow speed-independent solutions based on the local confirmation principle, i.e. though the solution exists, it does not satisfy the principle. For such examples TaxoSynthesis was not able to produce a solution without insertion of new signals, therefore they were not included in the table, as signal insertion is yet far not optimal in the TaxoSynthesis.

6 Conclusions

In this paper we have presented a new methodology for the synthesis of asynchronous circuits. The proposed approach utilizes the verification driven exploration of the solution space. The proposed methodology has the following advantages:

1. the technology basis can be flexibly accounted by the explicit weight function specification
2. all the produced circuits are verified to be correct by the synthesis algorithm, what eliminates the need for post verification.
3. processing time is significantly lower comparing to Petrify for large concurrent examples.

Preliminary results were found to be promising for synthesis of large asynchronous circuits. Future development is clearly required, in the areas of: 1) improving solution optimality by selecting of optimal mapping of Boolean functions into intervals’ set; 2) automated transformation of the implementable STGs into the canonical form; 3) decomposition. Another area of future research is increasing the synthesized circuits’ robustness by relaxing isochronous fork condition.

Example	TaxoSynth Processing Time (s)	Petrify Processing Time (s)	Processing time gain	TaxoSynth literals	Petrify literlas	Complexity gain
Chu150	0.95	1.16	1.22	13	11	0.85
Future	1,4	4,6	2,88	20	25	1.25
Intel div3	1.5	0.29	0.19	18	18	1
mp-froward-pkt	1.2	2.49	2.08	19	17	0.89
nak pak	1.15	3.04	2.64	18	20	1.1
nowick	0.92	1.39	1.51	16	14	0.88
ram read sbuf	1.4	4.32	3.09	26	22	0.85
roberto	0.9	1.97	2.19	15	15	1
sbuf-read-ctl	1.13	1.63	1.44	16	16	1
sbuf-send-ctl	1.12	2.88	2.57	21	19	0.9
sendr-done	0.67	0.38	0.57	5	5	1
vbe4a	0.81	0.95	1.17	8	8	1
vbe5b	0.84	1.2	1.43	13	14	1.08
vbe5c	0.81	1.03	1.27	10	11	1.1
trimos send extended	7.46	643.87	86.31	79	82	1.04
vbe10b extended	7.52	245.60	32.66	37	38	1.03

Table 1: Experimental results

References

- [1] Tam-Anh Chu. On the models for designing VLSI asynchronous digital circuits. *Integration, The VLSI journal*, 4(2):99–113, 1986.
- [2] L. Lavagno and A. Sangiovanni-Vincentelli. *Algorithms for synthesis and testing of asynchronous circuits*. Kluwer Academic Publishers, 1993.
- [3] J. Cortadella M. Kishinevsky A. Kondratyev L. Lavagno and A. Yakovlev. Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. In *IEICE Transactions on Information and Systems*, volume E80-D(3), pages 315–325, 1997.
- [4] A.Kondratyev J.Cortadella M.Kishinevsky E.Pastor O.Roig A.Yakovlev. Checking signal transtion graph implementability by symbolic bdd traversal. In *In Proc. European Design and Test Conference (ED&TC)*, pages 325–332, March 1995.
- [5] A. Semenov A. Yakovlev E. Pastor M. A. Pèna J. Cortadella. Synthesis of speed independent circuits from STG-unfolding segment. Technical Report TR565, University of Newcastle upon Tyne, England and Universitat Politècnica de Catalunya, Spain, 1996.
- [6] E.Pastor J.Cortadella A.Kondratyev O.Roig. Strtuctural methods for the synthesis of speed-independent circuits. In *In Proc. European Design and Test Conference (ED&TC)*, pages 340–347, March 1996.
- [7] N.A. Starodubtsev. Asynchronous processes and antitonic control circuits. *Soviet journal of Computer and System sciences (USA)*, English translation of *Izvestiya Akademii Nauk USSR. Tekhnicheskay Kibernetika (USSR)*. 1985,23(2).pp.112-119 Part I. Description language, 1985,23(6).pp.81-87 Part II. Basic properties, 1986,24(2).pp.44-51 Part III. Realization.
- [8] M.N.S.Swamy K.Thulasiraman. *Graphs, Networks, and Algorithms*. John Wiley & Sons, 1981.
- [9] A.Kondratyev M.Kishinevsky J.Cortadella L.Lavagno A.Yakovlev. Technology mapping of speed-independent circuits: decomposition and resynthesis. In *In Proc. International Symposium on Advanced Research in Asynchronous Cicuits and Systems, IEEE Computer society press*, April 1997.